International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

An Efficient Component Based Software Architecture Model Using Hybrid PSO – CS Algorithm

Prashanth Kumar Bolisetty[1*] , Prasanth Yalla[2]

[1]*Shinas College of Technology, Oman*
[2]*KL University, India*
*Corresponding author's Email: bprashanthkumar0779@gmail.com

Abstract: The Software architecture is generated by using the interfaces and structural components of the software systems in an organization. Software architecture, along with the structure and behavior, also concerned with functionality, performance, reuse, economic and technological constraints etc. In software its components are related to one another in large variety of ways. The main intension of our research is to build the component based software architecture with adaptive configurations using Particle Swarm Optimization and clustering techniques. Building architecture is an inspiring progression. In this paper, we will propose Hybrid Particle Swarm Optimization (PSO) – Cuckoo Search (CS) algorithm for developing an adaptive software architecture based on the Process control model. Initially, the components are selected based on testcases generated. After that, adaptive architecture will be built by using PSO - CS on the basis of clustering results. The architecture is built along with its functional requirements, responsibility and evaluation. The functional requirements are given as graphs of functional responsibilities where modifiability, efficiency and traceability are considered as the quality attributes. The proposed method produced solution with increased quality and better metric values.

Keywords: Component architecture; Reliability; Computational time; hybrid Particle swarm optimization; cuckoo search

1. Introduction

During the past decade, self-adaptation has progressively become a fundamental concern in the engineering of software systems. This helps to reduce the high costs of software maintenance and evolution and to regulate the satisfaction of functional and extra-functional requirements under varying conditions. Even with wide investigation of adaptation mechanisms in the engineering of dynamic software systems, their application to real problems is still limited due to lack of methods for validation and verification of complex, adaptive, nonlinear applications [1]. A major challenge in self-adaptive systems one has to face is to provide guarantees about the required runtime qualities. A self-adaptive system comprises of two parts: the managed system that deals with the domain functionality and the managing system that monitors the managed system. This is adapted to achieve particular quality objectives. The main underlying idea behind self-adaptation is complexity management through separation of concerns [2].

The Dynamic adaptive systems capabilities include automotive systems, telecommunication systems, environmental monitoring, and power grid management systems. It is to be enabled to tolerate a range of environmental conditions and contexts, but the exact nature of these contexts remains vague [3]. The need for adaptability arises more 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. An adaptation can be termed safe if (1) it does not violate dependency relationships and (2) if it does not interrupt communication either within a component or between components that would potentially yield erroneous or unexpected results. If adaptive software mechanisms are not grounded in formalisms that codify invariants and other properties that it must hold during decomposition,

the resulting systems will be prone to errant behavior [4]. Software architecture design is of supreme importance to the software development life-cycle and is used to represent and communicate the system structure and behavior to all of a system's stakeholders. In addition to this, architecture can facilitate stakeholders in understanding architecture design decisions and design rationale, further promoting a communication and understanding, reuse and efficient evolution [5].

Software architecture (SA) is given great importance to the software development life-cycle which is used to represent and communicate the system structure and behavior to all of its stakeholders with various concerns. Additionally, SA facilitates stakeholders to understand design decisions and rationale, thereby promoting reuse and efficient evolution. One of the major issues to be tackled in software systems development today is systematic SA restructuring to accommodate new requirements due to the new market opportunities, technologies, platforms and frameworks [6]. The ultimate goal of software engineering is to enable automatically produce software systems based on their requirements. At present, we pass the synthesis of executable programs, and concentrate on the automated derivation of architectural designs of software systems. This is made possible because architectural design largely means the application of known standard solutions in a combination that optimizes the quality properties of the software system [7].

The software architecture of a system is the set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both. The term also refers to documentation of a system's software architecture. Documenting software architecture facilitates communication between stakeholders, documents early decisions about high-level design, and allows the reuse of design components and patterns between projects [8]. Software programming is a hard design task, mainly due to the complexity involved in the process. Nowadays this complexity is increasing to levels in which reuse of previous software designs are very useful to short cut the development time [9].

The various benefits of the software architecting are as given below

- Architecting helps manage complexity.
- Architecting ensures architectural integrity.
- Architecting reduces maintenance costs.
- Architecting provides a basis for reuse [10].

The major design task in building enterprise applications is to design good software architecture. During recent years, the notion of software architecture has emerged as the appropriate level for dealing with software quality. One of the major issues in software systems development today is quality. A quality attribute is a nonfunctional characteristic of a component or a system [11]. Software must possess the qualities like Safety, Reliability, Availability, Cost, Maintainability, Performance or Response, Time, Energy consumption [12]. There are some recent attempts to establish software science as a foundation of software engineering. This may promote more analytical reasoning about software architecture, if it becomes popular. Software architectural design would benefit from analytical reasoning with scientific foundations. Importance of software architecture in the software design process is generally accepted among practitioners [13].

The rest of the paper is organized as follows: Section 2 gives a brief discussion on various recent researches done on the software design field. Section 3 explains about the proposed technique for software architecture using HPSO. Section 4 gives the detailed explanation about the results obtained and the section 5 concludes our proposed methodology.

2. Related Research

Architecture-based management approaches promote the use of architectural models. They serve as guidelines for various management functions. Some of the recent research works 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. N. M. Villegas *et al* [1] 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. G. Tamura *et al*. [14] 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. D. Weyns [2] 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. P. Tarvainen [15] 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. F. Berkes [16] 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.

3. Component Architecture for Software Adaption

If architecture can be termed as the set of plan resolutions, then texting the architecture reduces the documentation of the collection of plan resolutions. However, this does not appear to be practical. Our

general position is to get at the outcome of the plan resolutions, with the results selected, but the logic behind them is inaccessible. Most of the logic behind the solutions generally vanishes for good, or is stored only in the brains of the few people linked with them, if they are still around. Therefore, the logic behind a solution is not acquired definitely. But it is implied data, indispensable for the solution selected, but not recorded.

The data containing the pros and cons of software structural design and its background is called Architectural Knowledge and has caused a standard modification in the software architecture community. The most significant kind of AK is architectural decisions, which moulds software architecture. Supplementary kinds of AK is comprised of notions from architectural plan, needs engineering, people and the growth activity. Software architecture is normally the structure of modules in a program or system, their inter relations, and the doctrines and plan strategies. They help to manage the plan and growth in due course.

3.1. Proposed Methodology for Software Architecture

Our research work intends to build the component based software architecture with adaptive configurations using Particle Swarm Optimization and clustering techniques. Since Building architecture is an inspiring progression, in this paper, we will propose Hybrid Particle Swarm Optimization (PSO) – Cuckoo Search (CS) algorithm. This is used for developing an adaptive software architecture based on the Process control model. In the initial stage, the components selection is done based on test cases. Subsequently, adaptive architecture will be built using PSO - CS on the basis of clustering results. The architecture building is enabled with its functional requirements, responsibility and evaluation. The basic block diagram of our proposed component based software architecture model is shown in the figure 1 below:

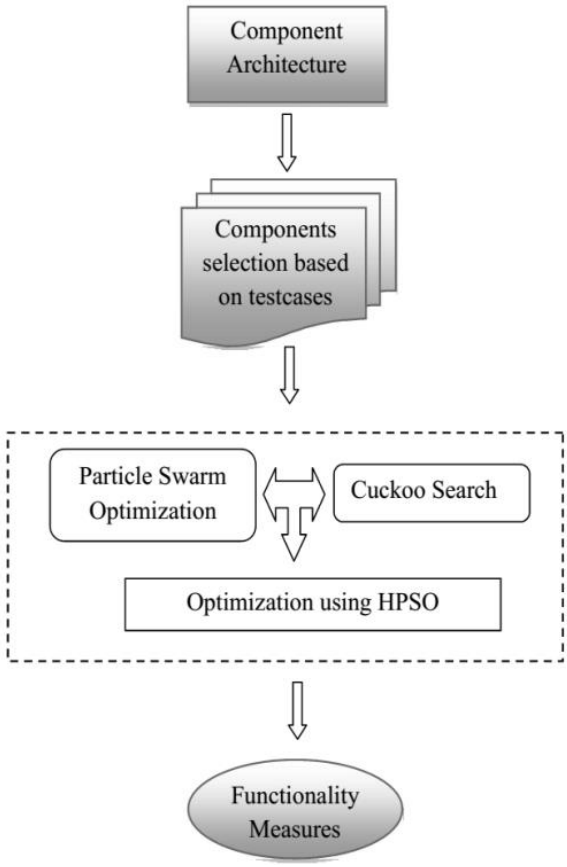

Figure.1 Flow diagram of our proposed method

3.2. Component Optimization with the Aid of HPSO

PSO designing was done on the basis of the social behavior of birds in a flock. In PSO, each particle soars in the exploration space with a velocity adjusted by its own flying memory and also by its companion's flying experience. It is a fitness function which determines the task value of each particle. PSO is an evolutionary algorithm which is very similar to that of the Genetic Algorithm. Here a particular scheme is initiated by considering a population of arbitrary solution. Here we have incorporated the Cuckoo Search algorithm into PSO which helps to obtain better optimization result than in normal PSO.

In HPSO along with each solution, arbitrary velocity is also assigned. This forms a particle which monitors its coordinates in the problem space in association with the best solution. For the remaining process to be executed, the fitness value remains the major consideration. This is referred to as pbest, whereas Gbest in HPSO refers to the location of the above solution.

The HPSO thus provides better solution and the steps involved are given in the below section. Hospital management application is the application

used in our proposed method. The test cases generated from this input is applied to the HPSO for optimization. It is through which the optimized components are selected.

3.2.1. Steps in Hybrid Particle Swarm Optimization

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

- First initiate a population of elements (solutions) with position and velocity chosen randomly for n-variable in the problem space.
- For each of these arbitrarily created elements estimate the fitness functions in n- variables.
- Now compare this fitness value with the particles pbest value. If these recent fitness value is improved than the pbest then chose the current fitness value as the pbest for the further processing.
- These fitness values is weighed against the overall finest preceding values and if the current value is better then update the gbest for the current particles array index and value as the new gbest.
- The position and velocity of the particle are assorted and then repeat the steps until the criterion of better fitness is obtained. Updation of velocity and the position of the particle is performed by utilizing the equations given below,

$$x_i(t+1) = x_i(t) + a_1 k_1 (d_i(t) - p_i(t)) + a_2 k_2 (d_i^{'}(t) - p_i(t)) \quad (1)$$

$$p_i(t+1) = p_i(t) + x_i(t+1) \quad (2)$$

Here 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,

$$d_i(t+1) = \begin{cases} d_i(t), & f(p_i(t+1)) \geq f(d_i(t)) \\ p_i(t+1), & f(p_i(t+1)) < f(d_i(t)) \end{cases} \quad (3)$$

The fitness of the solution is estimated using the equation mentioned above and the better solution are selected based on these fitness values. The obtained solutions from the PSO are then given as input to the cuckoo search algorithm inorder to optimize the solution further. The optimized solution from PSO will be the input for cuckoo which is further processed

3.2.2. Cuckoo Search Algorithm

The cuckoo search algorithm represents a biologically inspired algorithm. Its origin can be dated back to the breeding conduct of the cuckoos and it is easy to implement. Each egg signifies a solution and an egg of a cuckoo corresponds to a novel solution. The novel and superior solution replaces the worst solution in the nest. The various steps that are involved in the modified cuckoo search algorithm is explained briefly in the below steps,

Step 1: Initialization Phase

The population ($P_{i,}$ where i=1, 2, N) of host nest is initialized at random.

Step 2: Generating New Cuckoo Phase

With the help of the levy flights a cuckoo is selected randomly which generates novel solutions. Subsequently, the engendered cuckoo is evaluated by employing the objective function for ascertaining the excellence of the solutions.

Step 3: Fitness Evaluation Phase

The fitness function is evaluated in accordance with Equations 4 and 5 shown hereunder, followed by the selection of the best one.

$$F_m = \frac{P_C}{p_N} \quad (4)$$

$$fitness = \max imum\ popularity = F_m \quad (5)$$

Where,

P_C - signifies the selected population

P_N - represents the total population

Step 4: Updation Phase

At the beginning, the solution is optimized by the levy flights by employing the cosine transform. The superiority of the fresh solution is estimated and a nest is chosen at random among them. If the superiority of new solution in the elected nest is advanced to the previous solution, it is restored by the new solution (Cuckoo). Otherwise, the preceding solution is considered as the finest solution. The levy flights employed for the general cuckoo search algorithm is expressed by the Equation 6 shown below:

$$Lf_i^{*} = Lf_i^{(n+1)} = Lf_i^{(n)} + \alpha \oplus Lvy(N) \quad (6)$$

Step 5: Reject Worst Nest Phase

In this section, the worst nests are unobserved, considering their possibility values and fresh ones are created. Consequently, based upon their fitness function the best solutions are ranked. Thereafter, the best solutions are distinguished and marked as optimal solutions.

Step 6: Stopping Criterion Phase

The above process is repeated until the best solution is reached based on the termination criteria.

4. Result and Discussion

In our experimental procedure, 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. The hospital management system is utilized in our proposed method as the source software and the input to our proposed system is the hospital management application. From the Hospital management system, the test cases are extracted and the components are optimized. The test cases involved are from a number of derived classes. Our analyzed components are coupled with device, coupling of object, cost, reliability and popularity.

It is on the basis of the fitness value of the parameter chosen that the whole process labors in Hybrid Particle Swarm optimization. For the purpose of 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 the results are charted. While viewing Table 1 it can be confirmed that the fitness value for the suggested technique verified is superior to the technique where there is GA application.

Table 1. Fitness value between HPSO, Conventional PSO and GA for each iteration.

Iterations	Fitness value		
	Proposed method using HPSO	PSO	Using GA
5	21	15	16
10	19	13	15
15	17	13	13
20	15	12	13
25	13	12	13

For the above table the corresponding graphical representation is shown in the below figure 2. From the graph it is clear that our proposed method has delivered better outcome in terms of fitness when compared to conventional PSO and GA

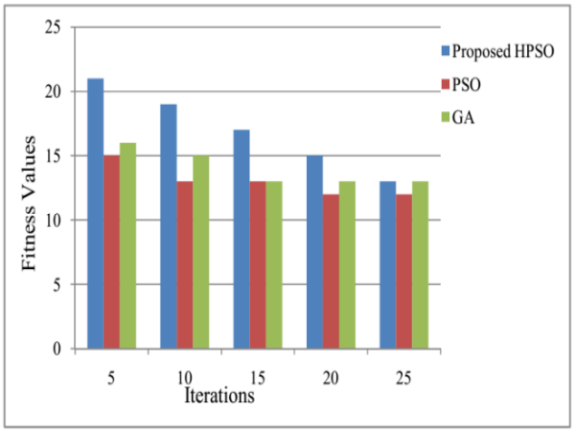

Figure.2 Comparison of Fitness value between HPSO, Conventional PSO and GA

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 HPSO are computed and the resulting values are charted. The similar evaluation for existing algorithms like PSO and GA are also tabulated inorder to compare our proposed system performance. The significances of the computation time we attained for different test cases are shown in Table 2.

Table 2. Computational time for HPSO, Conventional PSO and GA for various optimal testcases

Optimal Test Cases	Computational time		
	HPSO	PSO	GA
5	659	804	4329
10	951	1122	4531
15	1456	1781	4923
20	1821	2078	4986
25	2131	2603	5083

The graph is designed for computational time for optimal test cases based on the values shown in above table. Using HPSO, PSO and GA the graphical representation of computational time for our suggested method are compared which is shown in figure 3. As shown in the graph, the computational time for PSO has been less significant when match up to that of GA.

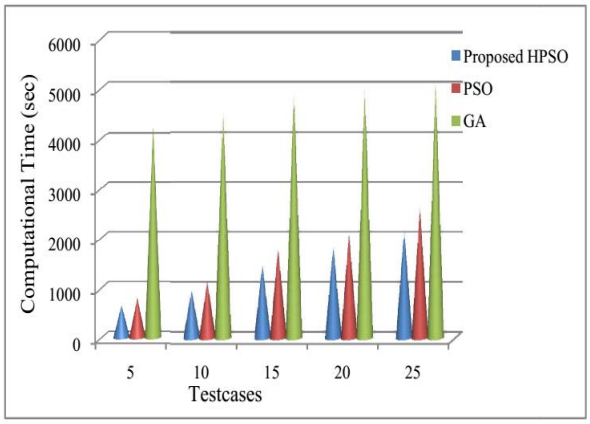

Figure.3 Comparison of Computational Time for HPSO, PSO and GA

5. Conclusion

In this paper, we have proposed Hybrid Particle Swarm Optimization (PSO) – Cuckoo Search (CS) algorithm for developing an adaptive software architecture based on the Process control model. Initially, the components were clustered based on an efficient clustering algorithm. After that, adaptive architecture would be built by using PSO - CS on the basis of clustering results. The architecture was built along with its functional requirements, responsibility and evaluation. The results showed that our proposed method had delivered better results in terms of test case optimization values when compared to other optimization techniques like conventional Particle Swarm Optimization (PSO) and Genetic Algorithm (GA). In future, we have planned to improve the computational time by employing alternate optimization techniques and also various other architectural parameters could be considered which can aid in better architecture of software.

Reference

[1] N. M. Villegas, G. Tamura, and R. Casallas, "A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems", In *Proc. of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 80-89, 2011.

[2] D. Weyns, "Towards an Integrated Approach for Validating Qualities of Self-Adaptive Systems", In *Proc. of the 10th International Workshop on Dynamic Analysis*, Minneapolis, MN, pp. 24-29, 2012.

[3] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Chengy and J. M. Bruel, "ELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems", In *Proc. of the 17th IEEE International Requirements Engineering Conference*, pp. 79-88, 2009.

[4] J. Zhang, B. H. C. Cheng, Z. Yang, and P. K. McKinley, "Enabling Safe Dynamic Component-Based Software Adaptation", *Architecting Dependable Systems III*, pp. 194–211, 2005.

[5] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. A. Babar, "A comparative study of architecture knowledge management tools", *The Journal of Systems and Software*, Vol. 88, No. 3, pp. 352-370, 2009.

[6] L. Dobrica, A. D. Ioniţa, R. Pietraru And A. Olteanu, "Automatic Transformation Of Software Architecture Models", *U.P.B. Sci. Bull, Series C*, Vol. 73, No. 3, pp. 3-16, 2011.

[7] O. Raiha, E. Makinen and T. Poranen, "Using Simulated Annealing for Producing Software Architectures", *Thesis, Department Of Computer Sciences, University Of Tampere*, pp. 2131-2136, Apr 2009.

[8] T. Ludik, J. Navratil, and A. Langerova, "Process Oriented Architecture for Emergency Scenarios in the Czech Republic", *World Academy of Science, Engineering and Technology*, Vol. 59, pp. 2342-2351, 2011.

[9] A. Sharma, R. Kumar, and P. S. Grover, "A Critical Survey of Reusability Aspects for Component-Based Systems", *World Academy of Science, Engineering and Technology*, Vol. 19, pp. 411-415, 2007.

[10] P. Eeles, "Software Architecture Masterclass", *In. proc. of IBM Rational Software Conference*, 2009.

[11] H. Gumuskaya, "Core Issues Affecting Software Architecture in Enterprise Projects", *World Academy of Science, Engineering and Technology*, Vol. 9, pp. 32-37, 2005.

[12] I. Meedeniya, "Robust Optimization of Automotive Software Architecture", *In. proc. of AutoCRC Technical Conference*, 2011.

[13] P. P. Dey, "Strongly Adequate Software Architecture", *World Academy of Science, Engineering and Technology*, Vol. 60, pp. 366-369, 2011.

[14] G. Tamura, N. M. Villegas, H. A. Muller, J. P. Sousa, B. Becker, G. Karsai, S. Mankovskii, M. Pezze, W. Schafer, L. Tahvildari, and K. Wong, "Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems", *Software Engineering for Self-Adaptive Systems*, pp. 108-132, Jun 2012.

[15] P. Tarvainen, "Adaptability Evaluation at Software Architecture Level", *The Open Software Engineering Journal*, Vol. 2, No. 1, pp. 1-30, 2008.

[16] F. Berkes, "Evolution of co-management: Role of knowledge generation, bridging organizations and social learning", *Journal of Environmental Management*, Vol. 90, No. 5, pp. 1692-1702, 2009.