COMPONENT EVALUATION FOR ADAPTIVE COMPONENT-BASED SOFTWARE ARCHITECTURE USING FUZZY LOGIC

Y. Mohana Roopa[1], Dr. A. Rama Mohan Reddy [2]

[1] Research Scholar, Dept of CSE, JNTU Anantapur, A. P. India
[2] Professor, Dept. of CSE, SVU College of Engineering, Tirupati, A.P., India

Corresponding author: Y. Mohana Roopa, mroopasvuphd@gmail.com

ABSTRACT: The component-based software engineering (CBSE) follows the process of reusability and reconfiguration of components to achieve the better productivity. The context-aware systems are part of CBSE, which monitors the behavior of the system and adopt automatically according to the execution context. In this paper, we are concentrating on the aware context policies that automatically adapt to the given context given by the clients and redesign the software architecture based on the requirements. The component repository was introduced, where it contains the number of reusable components. The fuzzy logic was applied to the component selection in the component repository. The GRASP algorithm is used to optimize the system architecture. The dish TV middleware is used to test the adaptability of the system.
KEYWORDS: Components, GRASP, Fuzzy adapter, Context, Ontology.

1. INTRODUCTION

The adaptive software architecture works on the component based development. The components are the building parts by combining the software coding [GC92. ZJB06]. Software adaptation is the process of changes the component functionality according to the user requirements dynamically. It includes the detection of interaction failure and their repair when it is required [D+13, HJ13, S+12]. Adaptation is considered in various aspects like from the component point of view or service point of view. If it is from the component side, the actors are interfaces, services, and ports, and if it is from service point the actors are interfaces and services.
As per the researchers wise, the software developers can combine software components from any commercial providers (COTS). Therefore, there is no restriction for combining the components from different models. It means the developers have the flexibility to choose the components from different providers. Assume that there is a number of different service models and components, the software architect needs to choose a language that is suitable for component development for appropriate adaptation. Moreover, component models that are not restricted to simple call and response; the component contains the component interface to provide the interactive response and support adaptation facilities which increases the component reusability.
In this paper, we address the component evaluation for increasing the reusability of the component. The component repository is introduced to increase the component interaction internally. We investigate the fuzzy logic application into the model for component adaptation.
This paper is organized as follows. Section 2 describes the related work in the field of component-based software adaptation. Section 3 deals with component evaluation for the adaptive component model based on the fuzzy logic implementation. In section 4, we consider the optimization model for the component-based system using cost function. Section 5 deals with GRASP algorithm for optimization. In section 6, the experimental evaluation is performed, and finally, Section 7 concludes with perspectives on work.

2. LITERATURE SURVEY

In [DF09], the authors proposed a KPN models for designing the model- controller- adapter framework. This framework is utilized for self-adaptivity of the streaming applications. The paper did not discuss the details about the monitor and adaptation controllers. Despite the fact that a contextual investigation is given [DGV14], no outcomes are accessible to evaluate the framework regarding its adequacy and execution (convergence, timing and so forth.).
The standard mechanism for self-adaptivity is provided in [CL03, RKR14], which separates the concerns between system functionalities and self-management. Self-adaptivity is achieved by applying an arrangement of adaptation strategies on programming segments while certain configurable framework triggers these approaches. Conceivable adaptations for part conduct and application parameters are additionally examined. Unfortunately, they don't discuss if and how a general objective is accomplished.

In [G+09], the authors exhibit the examination results of the project MADAM that has conveyed a thorough answer for the advancement and operation of self-adaptive applications and context-aware service. The fundamental commitments of this work are (a) a complex middleware that backings the dynamic adaptation of component-based models, and (b) a creative model-driven improvement approach taking into account abstract adaptation models and relating model-to-code changes.

A model of a self-adaptable framework is introduced in [DFT09], in which a proposition to oversee self-adaptivity at hardware and programming levels by the method for a decentralized control algorithms is given. An objective methodology, an objective particular interface, alongside a decentralized and facilitated control system is proposed as a major aspect of this work.

A middleware-based model to deal with run-time swapping of procedures among tiles of a NoC is introduced in [C+05]. Such a system helps in accomplishing adaptivity support, for example, adaptation to non-critical failures.

In [C+07, J+08, E+12], different methods for fine-grained QoS control of multimedia applications are exhibited. The proposed methods create a controlled application that meets given QoS prerequisites from a data application programming. The monitoring mechanism monitors the advancement of the algorithm in a cycle and picks the following activity to run and its quality level, guided by security and optimality requirements for the framework.

3. COMPONENT EVALUATION FOR SELF-ADAPTIVE COMPONENT ARCHITECTURE

Figure 1 shows the architecture for component evaluation. For evaluating the given component, the evaluator is used. The evaluator works with the principle of fuzzy logic and it checks the adaptability of the given component with the context [HH11]. The adaptability of the component ranges from 0 % to 100%. Let us consider that, there is a request for component evaluation, the evaluator module sends the request for collecting information from the storage manager, and the information contains the set of rules. The format of the rules is given below

```
If ( Condition )
 {
   " component is adequate."
Else
" component is not adequate."
```

By using above rule, one can simplify that it is better to use first order logic for deciding the component is adequate or not.

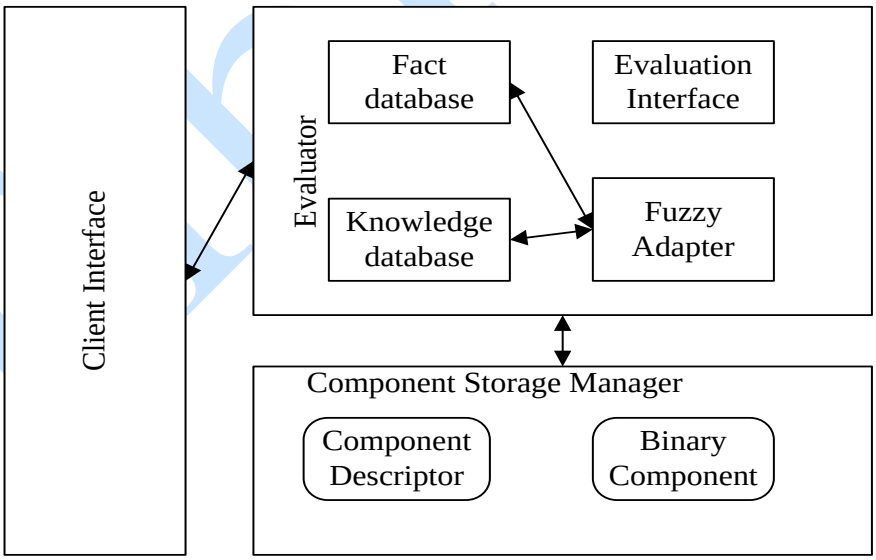

Figure 1: Component Evaluation Architecture

For the generated rules, we can apply fuzzy logic to find out that the component is much, moderate or less adequate which was well explained in [W+12]. The evaluator module contains the knowledge database that stores the component description. The fact database contains the context data that was sent by the client interface for execution. The fuzzy adapter is introduced in evaluator module that contains the following parameters.

i. The fuzzy set contains the suitable functions that the input function may belong.
ii. The fuzzy logic contains the mathematical operations

iii. The knowledge base is associated with fuzzy rules.
iv. Fuzzy sets contain the output variables of the inference engine.

The context environment contains the ontology that is utilized to characterize the significant concepts to the domain and the range of these concepts (the client interface monitors the concepts). The fuzzy adapter manages the adequacy level of the components. The component descriptor contains component parameters that are associated with component storage. The component developer can understand the evaluator module by examining the component description [K+12, KY95]. The detailed explanation Fuzzy adapter is shown in figure 2. The framework in figure 2 examines the software self-adaptation on the context environment. The fuzzy adapter is connected with application logic through architecture designer. The application logic contains the reusable components that are associated with context environment. According to the functional requirements, the architecture designer selected the components from the application logic and evaluated by using the fuzzy adapter, the fuzzy adapter contains adaptation objectives such as rules, fuzzification, defuzzification and fuzzy inference.

The above component evaluation method causes some problems while evaluating the component for different manufacturers. For example, a company has specified that for evaluation of component is 60 on the concept and another company had specified it as 70 for the same context. In According to solve this problem the component descriptor is defined as well set of rules.

4. COST OPTIMIZATION FOR COMPONENT REPOSITORY

The proposed component repository contains the number of components which are needed to be connected to component-based architecture. The number of components in the component storage is high means the number of possible combinations is also high [Noe12]. For finding the adequate component architecture, we are developing a component optimizer based on the context sent by the client.

A.Problem formulation

A component based system is composed of required interfaces and provided interfaces. In our approach, both required the internal components use interfaces and provided interfaces in the component repository.

The component-based system contains the interconnected components with the combination of required interfaces and system provided interfaces. To optimize the component based system the following set of rules must satisfy.

i) The system provided interfaces are a must and should associate with provided interfaces of the component.
ii) The system provided interfaces are connected with 0 or maximum connections with required interfaces.
iii) The component required interfaces are connected with the corresponding component that is compatible.

By considering above set of rules the cost function is defined as follows:

.

A.1. Cost function

$$\alpha(y) = 100 - \omega(y) \quad (1)$$

where α(y) represents the total cost function of the system and ω(y) represents the evaluation of the component. By using the equation 1, we can calculate the cost of the individual component, the total cost of the system is calculated by the sum of each component cost. To achieve an optimal solution, the total cost of the system must be minimized.

To reduce the total cost of the software system, the mathematical model was developed. To maintain the properties of the component based model, we are introducing the mathematical model into the cost function.

$$\min(f) = \sum_{i \in CI} \eta_i \times \mu_i \quad (2)$$

where η is the cost when using the component i and μ be the decision variable if it is 1 means the component i is used in the current configuration; 0 otherwise. In equation 2, to achieve the optimal set of components to the given context, it is compulsory to minimize the objective function. The constraints of the cost function are given in equation 3, 4 and 5.

$$\beta_S - \delta_{kl} \le 0 \ \ where\ s = ikjl,\ i, j \in \mathrm{E}, k \in \mathrm{N}_i, l \in \mathrm{T}_j \quad (3)$$

In equation 3, where β_s be the decision variable, if the required interface k of component i is connected to the provided interface l of component j then it is set to 1, otherwise 0. E is the set of components in the component repository, N be the set of required interfaces and T be the set of provided interfaces. If the interfaces are not compatible means δ set to 0.

$$(\beta_s \times \delta_{kl}) - \chi_i = 0 \quad (4)$$

In equation 4, the component is used by the system means all set of required interfaces is connected to the single provided interface.

$$(\beta_s \times \delta_{kl}) - \varphi_{jl} \times \chi_i = 0 \quad (5)$$

where φ_{jl} is the maximum number of connections connected to the component j with the provided interface l. In equation 5, the provided interface l is used when only the component j is used in the system.

5. GRASP HEURISTIC ALGORITHM FOR OPTIMAL COMPONENT ARCHITECTURE

The Greedy Randomized Adaptive Search Procedure (GRASP) [Res01] approach is suitable to find the optimal solution for NP-Complete problems. For optimization of component architecture, we are employing the GRASP approach for improving the component accessibility and reliability regarding real system.

The GRASP contains the iterative algorithm that is composed of two steps. One is build, and another one is search. After successive iterations, the algorithms find the best solution based on the obtained solutions in iterations.

Build Stage: This stage is initiated with an empty solution set sol{}. In each iteration of the algorithm, y is selected from the set N and all components from E. The components that are compatible with the list N is placed on the list L and some of the optimal candidates are selected from the list L and placed in optimal list OL.

Search stage: This stage is considered with change and removes functions. To generate the alternate solution for the component set found in build stage, we use above mentioned two functions. The change function makes the changes to the solution set by changing the component. The remove function deletes the component from the solution.

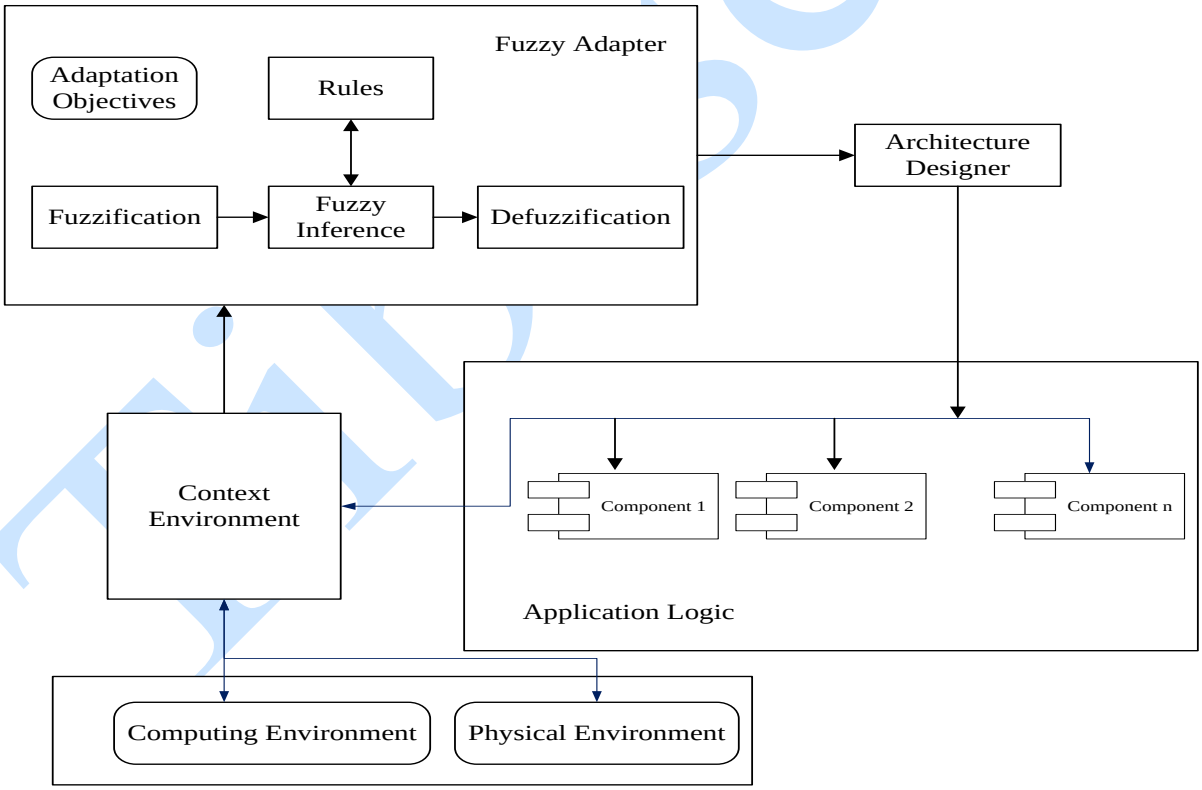

Figure 2: SaaS Cloud Reference Architecture

Algorithm 1: GRASP Algorithm

Begin
Step 1: Initialize N, T, E
Sol = {ϕ}
Step 2: while (N! = 0) do
{
y = initial (N)
if (exists(i) in T)
{
Connect y to its compatible interface in E
Continue
}
Step 3:L = { y in E }

$$\alpha_{\min} = \min\{\alpha(y) of\ L\}$$

$$\alpha_{\max} = \max\{\alpha(y) of\ L\}$$

Step 4:OL = {all y in L }

$$\alpha(y) \le \alpha_{\min} + y(\alpha_{\max} - \alpha_{\min})$$

Step 5: select the random component z from OL
Step 6 : sol = sol + {z}
}
Return sol
End

Algorithm 1: Greedy Randomized Adaptive Search Procedure

5. EXPERIMENTAL DESIGN

The implementation of the proposed component evaluation model is done in the middleware of the Dish TV. The test is conducted on the set-top box receiving the digital signal through the modem. To observe the test results, we apply the fuzzy logic by describing two input functions called as system resolution and video resolution. To evaluate the results we set the parameter as low resolution, standard resolution, and high resolution. The deciding factor would range from 0 to 100. If the video resolution obtained 100 means, it is more adequate.

We have considered the three development techniques for the component to validate the fuzzy logic engine. Along, with the component development, the system considers the five different contexts on the adaptation mechanism.

After achieving the adaptation we, are considered the building stage of the component model, it considers the GRASP algorithm with Metaheuristics. The number of components in the set E is taken as {20, 30, 40, 50, and 60}. After complete execution of an algorithm, the number of randomly provided interfaces and required interfaces are generated.

The algorithm was written in Java language to support the component development in a flexible manner. The results are shown in Table 1.

Table 1: GRASP algorithm evaluation

Number of Components	Branch and Bound		GRASP	
	Time (s)	Optimal solution (O)	Time (s)	O_{avg}
20	2	765	0.056	765
30	5	427	0.042	458
40	14	634	0.06	715
50	25	523	0.057	546
60	34	650	0.124	684

The branch and bound method is tested with a number of components, the time to reach the solution are mentioned in Table 1, and the optimal solution O found by the method is given. The GRASP algorithm is tested with a number of components used by the branch and bound method. The time taken to achieve the solution is less when compared to the branch and bound, and O_{avg} is the optimal solution found after every five executions by applying metaheuristics. The LINGO optimized model is used to obtained the results of the branch and bound method [***12a]. To implement the middleware software in the dish TV, the system utilizes the OWL representation using the Jena framework [***12b]. The fuzzy interface engine is implemented using Mat lab mathematical tool [***11].

6. CONCLUSION

This paper presents a new approach for component evaluation for adaptive component based software system. We define the context aware policies to execute the component based on the functionality. The approach follows the fuzzy logic to decide which components are suitable for the given context. The model proposed an optimized approach for finding an adequate component using GRASP Metaheuristic algorithm. To achieve the concrete results, the model is implemented under a case study by developing middleware for the dish TV. The results obtained for the implementation is convincing, and the implementation of reconfiguration procedures on the client side and server side is satisfactory.

REFERENCES

[CL03] **P. Charles David, T. Ledoux** - *Towards a framework for self-adaptive component-based applications*, in Proceedings of Distributed Applications and Interoperable Systems, vol. 2893 of Lecture Notes in Computer Science, pp. 1–14, Springer, 2003.

[C+05] **J. Combaz, J. C. Fernandez, T. Lepley, J. Sifakis** - *Fine grain QoS control for multimedia application software*, in Proceedings of the Design, Automation and Test in Europe (DATE '05), vol. 2, pp. 1038–1043, March 2005.

[C+07] **J. Combaz, J. C. Fernandez, J. Sifakis, L. Strus** - *Using speed diagrams for symbolic quality management*, in Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS '07), pp. 1–8, March 2007.

[DF09] **O. Derin, A. Ferrante** - *Enabling self-adaptivity in component based streaming applications*, in SIGBED Review Special Issue on the 2nd International Workshop on Adaptive and Reconfigurable Embedded Systems (APRES '09), vol. 6, 2009.

[DFT09] **O. Derin, A. Ferrante, A. V. Taddeo** - *Coordinated management of hardware and software self-adaptivity*, Journal of Systems Architecture, vol. 55, no. 3, pp. 170–179, 2009.

[DGV14] **L. D. Dhinesh Babu, Angappa Gunasekaran, P Venkata Krishna** - *A Decision Based Pre-emptive Fair Scheduling Strategy to Process Cloud Computing Work-flows for Sustainable Enterprise Management*, International Journal of Business Information Systems, Vol. 16, No.4, pp. 409-430, Inderscience Publishers, 2014.

[D+13] **Rogério De Lemos et al.** - *Software engineering for self-adaptive systems: A second research roadmap*. Software Engineering for Self-Adaptive Systems II. Springer Berlin Heidelberg, 2013. 1-32.

[E+12] **E. Cannella, O. Derin, P. Meloni, G. Tuveri, T. Stefanov** - *Adaptivity support for mpsocs based on process migration in polyhedral process networks*, VLSI Design, vol. 2012, Article ID 987209, 17 pages, 2012.

[GC92] **D. Gelernter, N. Carriero** - *Coordination Languages and their Significance*, Commun. ACM 35 (1992), p. 96.

[G+09] **K. Geihs, P. Barone, F. Eliassen et al.** - *A comprehensive solution for application-level adaptation*, Software – Practice and Experience, vol. 39, no. 4, pp. 385–422, 2009.

[HH11] **Chao-Jung Hsu, Chin-Yu Huang** - *An adaptive reliability analysis using path testing for complex component-based software systems*. Reliability, IEEE Transactions on 60.1 (2011): 158-170.

[HJ13] **Jim Highsmith** - *Adaptive software development: a collaborative approach to managing complex systems*. Addison-Wesley, 2013.

[J+08] **M. Jaber, J. Combaz, L. Stras, J. C. Fernandez** - *Using neural networks for quality management*, in Proceedings of the 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '08), pp. 1441–1448, September 2008.

[KY95] **G. Klir, B. Yuan** - *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Englewood Cliffs, NJ, USA, 1995.

[K+12] **Narges Khakpour et al.** - *Formal modeling of evolving self-adaptive systems*. Science of Computer Programming 78.1 (2012): 3-26.

[Noe12] **Victor Noël** - *Component-based software architectures and multi-agent systems: mutual and complementary contributions for supporting software development*. Diss. Université de Toulouse, Université Toulouse III-Paul Sabatier, 2012.

[Res01] **M. Resende** - *Greedy randomized adaptive search procedures (GRASP)*, in Encyclopedia of Optimization, vol. 2, pp. 373– 382, Kluwer Academic Publisher, 2001.

[RKR14] **T. S. K. Reddy, P. V. Krishna, P. C. Reddy** - *Power Aware Framework for Scheduling Tasks in Grid based Workflows*, Int. J. Communication Networks and Distributed Systems, Inderscience Publishers, 2014.

[S+12] **Lionel Seinturier et al.** - *A component-based middleware platform for reconfigurable service-oriented architectures*. Software: Practice and Experience 42.5 (2012): 559-583.

[W+12] **Danny Weyns et al.** - *A survey of formal methods in self-adaptive systems*. Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering. ACM, 2012.

[ZJB06] **Ji Zhang, Betty H. C. Cheng** - *Model-based development of dynamically adaptive software*. Proceedings of the 28th international conference on Software engineering. ACM, 2006.

[***11] *** - *MathWorks: Matlab — The Language of Technical Computing*" 2011, http://www.mathworks.com/products/matlab/

[***12a] *** - *Lindo Systems: Lingo— Optimization Modeling Software for Linear, non-linear and Integer Programming*, 2012, http://www.lindo.com/index.php?option=com content&view=article&id=2&Itemid=10.

[***12b] *** - *The Apache Software Foundation: Apache Jena*, 2012, http:// incubator.apache.org/jena/.