A Reusable Adaptation Component Design for Learning-Based Self-Adaptive Systems

Kishan Kumar Ganguly, Kazi Sakib
Institute of Information Technology
University of Dhaka, Dhaka, Bangladesh
Emails: bsse0505@iit.du.ac.bd, sakib@iit.du.ac.bd

***Abstract*—In self-adaptive systems, according to the separation of concern principle, the adaptation logic and the business logic components should be kept apart for reusability. However, this promotes reuse of the whole adaptation component while reuse of its subcomponents and their classes can also be helpful. Existing techniques do not consider this. Moreover, existing approaches also do not consider application and environment factors together for a more accurate adaptation. In this paper, a learning-based adaptation component design has been proposed which supports these. Machine learning is used to express metrics that measure system goals, as a combination of application and environment attributes. These are used to select application components to turn on or off by solving an optimization problem, aimed at maximizing system goal conformance. Components are turned on or off using a customizable effector component. Design patterns are utilized for increasing the reusability of the adaptation subcomponents. The proposed method was validated using the popular Znn.com problem. The reusability and learning accuracy metrics used indicate that it performs well for both. The system was also put under high load for observing adaptation of response time. It was seen that adaptation occurred as soon as the response time was over a provided threshold.**

Keywords–Reusable Adaptation Component; Environment Feature; Application Feature; Design Pattern.

I. INTRODUCTION

For self-adaptive systems, developing the business logic and then, augmenting it with the adaptation logic are easier due to the adaptation component complexity. Apart from this component-level reuse, subcomponent-level reuse (i.e., reuse of adaptation subcomponents and their classes) can further reduce development time. For this, the adaptation component needs to be customizable to easily add or remove any classes. Moreover, adaptation effectiveness should be ensured for maximum goal conformance (e.g., performance, cost, etc.) [1].

In self-adaptive systems, goals are generally non-functional requirements. These requirements are expressed using metrics (e.g., response time, throughput, etc.), which help to detect goal violation by checking metric thresholds. Goal violation leads to adaptation which triggers reconfiguration to toggle (turn on or off) components. These components, also called *features*, are variation points of the system [2]. For example, modules for turning on and off a server can be called separate features. Generally, adaptation logic is a mathematical model that provides a *feature selection* to toggle. In the proposed methodology, these features, which can be toggled are called *application features*. However, *environment features* may exist that have impact on adaptation (e.g., service time, bandwidth, etc.) but cannot be toggled. The challenge is to incorporate these two types of features for effective adaptation and structuring the adaptation logic modularly for reusability.

A number of design techniques for self-adaptive systems have been proposed where a few seem to have considered reusability. Garlan et al. proposed the Rainbow framework where adaptation condition-action rules were hardwired into the system which hampered reuse [1]. Esfahani et al. proposed the FUSION framework, which used learning to derive equations for predicting metrics and used these to construct an optimization problem. This was solved to get a feature selection. However, incorporating environment features in the optimization problem leads to a feature selection that provides specific numerical values for the environment features. This is not useful because environment features cannot be controlled or selected, rather these depend on the underlying system environment. So, environment features cannot be directly used with the FUSION framework. Ramirez et al. discussed twelve design patterns for self-adaptive systems [3]. However, breaking these down to lower level patterns can facilitate reuse [4].

The contributions of this work are: 1) A generic design for the adaptation component that supports both component and subcomponent-level reuse. 2) A learning-based adaptation technique that considers environment features and generates training data automatically. The proposed approach applies machine learning to derive feature-metric equations to predict metric values from application feature statuses (on or off) and environment feature values. These more accurate metric equations are combined with the user provided metric thresholds to construct utility functions. These are used to devise an integer linear optimization problem similar to FUSION in case of goal violations. However, unlike FUSION, the environment features are also considered. The feature selection given by solving the optimization problem is applied to the system using components called *customizable effectors*. The proposed methodology also introduces a technique to automatically derive the data for learning. All these make the adaptation logic generic, which helps to reuse the component as a whole. For subcomponent-level reuse, design patterns are utilized to structure the adaptation component modularly.

The proposed technique was applied to the Znn.com model problem [1]. This system was deployed in five servers with a load balancer. Reusability was assessed using renowned metrics (e.g., Afferent Coupling, Rate of Component Observability, etc.). Effectiveness was validated by observing whether the response time stays under an empirically derived threshold in a high load environment. The reusability metric values indicate higher reusability in both component and subcomponent-level. The proposed technique also performs better in the high load as it brings down the response time once it rises. Moreover, learning accuracy metrics (e.g., Adjusted R^2, Correlation Coefficient, etc.) were used to show that considering environment feature produces more accurate metric equations.

The rest of the paper is structured as follows. In Section II, the proposed reusable adaptation component design is presented. In Section III, a case study is provided along with

 ISBN: 978-1-61208-590-6

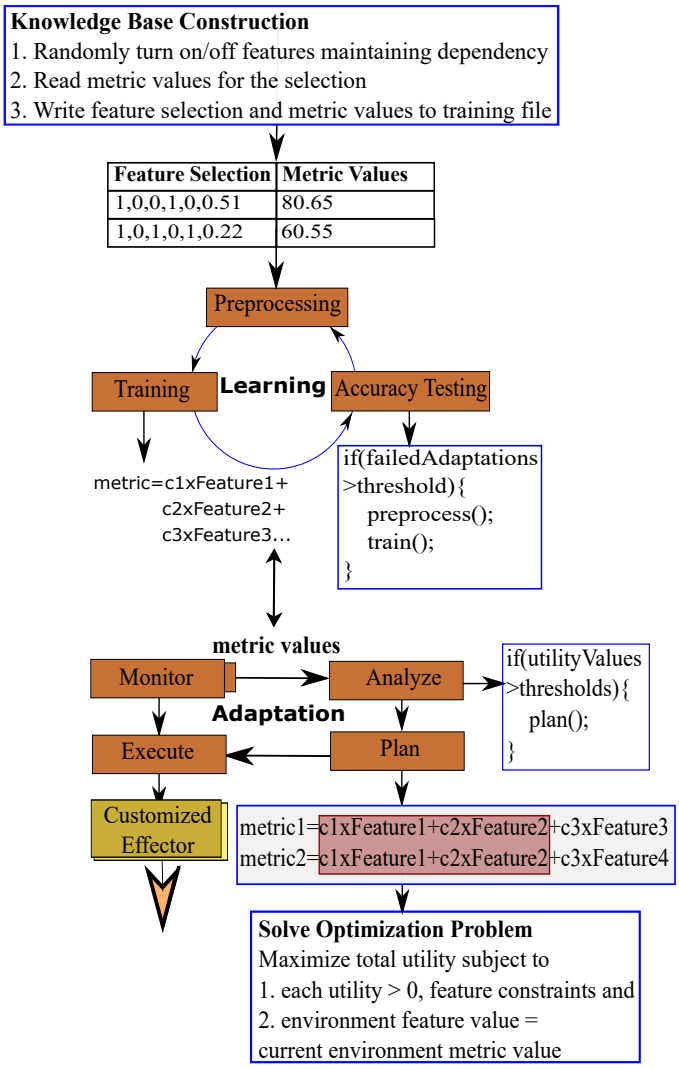

Figure 1. The Logical View of the Proposed Methodology.

an evaluation of reusability and effectiveness of the proposed approach. Section IV contains the related works. Section V holds the conclusion and future research directions in this area.

II. Reusable Adaptation Component Design

Here, a generic adaptation logic has been designed based on learning. Although this ensures reusability of the whole adaptation component, reusability of the subcomponents (for example, learning, preprocessing algorithm, etc.) is not guaranteed. For this, the subcomponents are structured with design patterns. These two perspectives are discussed below.

A. Logical View

The adaptation logic consists of three processes - Knowledge Base Construction (KBC), Learning and Adaptation. These three processes and the required system specific inputs are depicted in Figure 1 and discussed below.

1) Input: Information about application and environment feature, feature dependency, metric, utility and initial feature selection are required where application feature information consists of feature name only.

For feature dependency, the dependent features and their types are needed. The proposed method uses the dependency

TABLE I. Constraints for Feature Relationships

Feature Constraint	Feature Relation
$\sum_{\forall f_n \in \text{zero-or-one-of-group}} f_n \leq 1$	zero-or-one-of-group
$\sum_{\forall f_n \in \text{exactly-one-of-group}} f_n = 1$	exactly-one-of-group
$\sum_{\forall f_n \in \text{at-least-one-of-group}} f_n \geq 1$	at-least-one-of-group
$\sum_{\forall f_n \in \text{zero-or-all-of-group}} f_n \bmod n = 0$	zero-or-all-of-group
$\forall$child $\in$ Conflicting Feature Set $f_{parent} - f_{child} \geq 0$	parent child relation

types mentioned by Esfahani et al. [2] (Table I) because these cover common feature relationships and can be represented mathematically for the optimization problem. Here, *zero-or-one-of-group* means more than one feature cannot be enabled. *Exactly-one-of-group* means exactly one feature can be enabled at a time. *At-least-one-of-group* means at least one of the features must be enabled. *Zero-or-all-of-group* indicates either all or none of the features can be turned on. *Parent child relation* means enabling a specific (parent) feature requires all other features of the group to be enabled.

The existing system needs to expose an API for metric calculation. The metric information contains metric names, types, thresholds and API location (e.g., URL, class file path etc.). Two types of metrics are used representing maximization and minimization goals. For maximization goals, the metric values must be greater than the thresholds and the opposite for minimization goals. Metric types are used to form the utility equations using (1).

$$u_n = \begin{cases} m_n - th_n & \text{if } Type_n = Maximization \\ th_n - m_n & \text{otherwise} \end{cases} \quad (1)$$

Where u_n and m_n represent the utility and metric values respectively. th_n is the threshold value for the nth metric.

The initial feature selection is the feature selection for the first run. This is used in KBC. The environment features are also given. These features must have corresponding metrics provided in the aforementioned metric information. The metrics calculate current values for these environment features. In the optimization problem, these current values are considered for better accuracy of the solution.

2) KBC: This component generates training data for the learning process. As seen from Figure 1, training data consists of feature combination and metric values. Environment features generally have numeric values. So, the number of possible feature combination is infinite and cannot be generated. So, the application is put under a simulated or real environment and application features are toggled randomly (Figure 1). The environment feature values and metric values are read from the Monitor process and all the feature-metric values are written as training data. For example, in Figure 1, the random application feature selection is 1, 0, 0, 1, 0 for the first row and 0.51 is the environment feature value. Here, 1 and 0 indicates application feature status (enabled or disabled). This feature selection results in metric value 80.65. All these are written as the training data.

Features in each of the feature dependency groups are randomly toggled maintaining the dependency. For example, in case of at-least-one-of group, when one feature is randomly selected to turn on, all the other features are turned off. For parent-child relation, a number between 0 and 1 is chosen to

 ISBN: 978-1-61208-590-6

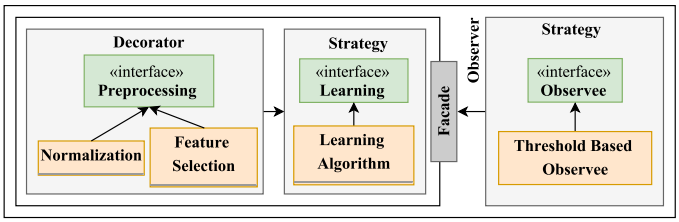

Figure 2. The Structural View of the Learning Component.

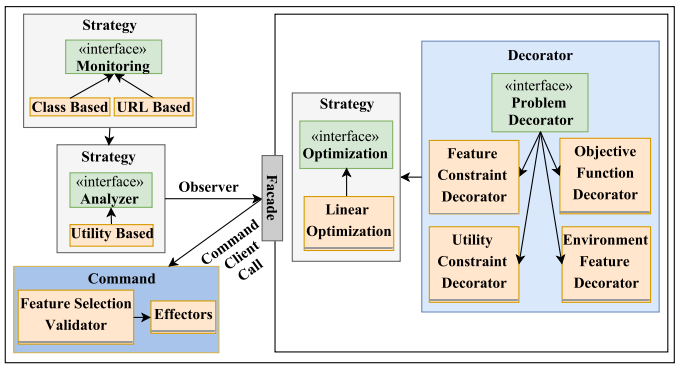

Figure 3. The Structural View of the Adaptation Component.

toggle the parent feature and all the child features are turned off or on accordingly. This random sampling with dependency groups ensures that feature dependency is maintained and the generated training data represents the population of training data appropriately.

3) Learning: Learning process aims to generate equations that predict metric values from feature selection. These equations are used in the Plan process. The generated training data from KBC is preprocessed to be properly used in the learning algorithm. For example, training data can be normalized for scaling. Preprocessing methods depend on the training data and the learning algorithm. So, it needs to be customizable. Strategy pattern is used for this purpose (Section II-B).

The next step is training where the preprocessed data is passed to a learning algorithm to derive metric equations. These equations help to predict metric values provided application and environment feature values. Training can be done using a regression algorithm. For example, in Figure 1, the metric equation from training is a linear regression equation.

As adaptation process will show, the training data is gradually updated with monitored metric values and feature selection. However, as the adaptation decision is taken using metric equations derived from previous training data, this can lead to failed adaptation when new patterns of data arrive. In this case, training is rerun when the number of failed adaptations exceed an empirically defined threshold (Figure 1).

4) Adaptation: Adaptation process consists of Monitor, Analyze, Plan and Execute components following the MAPE-K approach [5]. This process detects violated goals using the utility equations and solves an optimization problem to find a feature selection with maximum total utility function value. Although this technique closely resembles FUSION [2], the environment features are incorporated for better adaptation, which is one of the contributions of this work. The four components of Adaptation are discussed below.

a) Monitor: This collects metric values from the system using the metric API. These are stored in the knowledge base along with the current feature selection as training data.

b) Analyze: The metric values from Monitor are used in the utility equations for goal violation detection. From Equation (1), goal violations lead to $u_n < 1$. This is used to detect goal violations in this technique.

c) Plan: Detection of goal violation invokes Plan component. The metric equations from the learning process are used to find conflicting goals. The conflicting goal detection mechanism has been shown in Figure 1. The violated goal metric equation is matched with other metric equations to find overlapping features (shaded area in Figure 1). If overlapping features are present, these metrics are conflicting to the violated metric and these need to be considered together for optimization. Then, an optimization problem is formed.

$$F_{selection} = maximize(\sum_{i=1}^{n_c} U_i(M_i(F)))$$

Subject To

$$\forall i \leq n_c . U_i(M_i(F)) > 0$$
$$\wedge \forall f \in F . F_d(f)$$
$$\wedge \forall f_e \in F_e . f_e = c$$

Where

$$\forall i \leq n_c, M_i(F) = \sum c \times f \quad (2)$$

Here, $F_{selection}$ is the feature selection after solving the optimization problem. This feature selection contains all the feature values (0 or 1) to toggle. The optimization problem states that the total utility for n_c conflicting goals needs to be maximized. The constraints show that all utility functions in the maximization function must be greater than zero because individual goals must not be violated. Besides, feature dependencies must be maintained (i.e., F_d must be true). All the environment features f_e from the environment feature set F_e will have corresponding environment metric values. All The metrics $M_i(F)$ in the utility equation will be replaced by metric equations from the learning process.

d) Execute: Execute component helps to toggle selected features in the existing system. This component consists of some effectors which are used to toggle each of the features. These effectors are specific to the system and so, and abstractions are provided for later customization.

B. Structural View of The Model

In the structural view, the subcomponents of Learning and Adaptation have been organized with Gang of Four (GoF) design patterns [6] for reusability and customization. These were chosen by comparing the functionality of the subcomponents with the applicability of the design patterns [6].

Figure 2 and 3 show the design patterns for the Learning and the Adaptation components. *Decorator pattern* is used to provide additional functionality at runtime. In preprocessing, the training data is dynamically filtered with algorithms such as normalization, feature selection, etc., using this pattern (Figure 2). In optimization problem construction, decorators that add the feature, utility and environment feature constraints, and the objective function, build a complete optimization problem (Figure 3). *Strategy pattern* helps to support interchangeable algorithms. So, it has been used to support different learning algorithm and failed adaptation testing strategies in Learning.

 ISBN: 978-1-61208-590-6

TABLE II. DEFINITIONS AND THRESHOLDS FOR REUSABILITY METRICS

Metric Name	Level	Definition	Range/ Value
RCO	Component	Rate of Component Observability	[0.17, 0.42]
RCC	Component	Rate of Component Customizability	[0.17, 0.34]
SCCr	Component	Self-Completeness of Components Return Value	[0.61, 1.0]
SCCp	Component	Self-Completeness of Components Parameter	[0.42, 0.77]
LCOM4	Class	Lack of Cohesion of Methods 4	1
DIT	Class	Depth of Inheritance Tree	2
AC	Class	Afferent Coupling	[0,1]
WMC	Class	Weighted Methods per Class	[0,24]

In Adaptation, it has been used to support different algorithms for monitoring, analyzing goal violation and optimization.

Observer pattern helps to notify all the dependent objects. This has been used to notify the learning process to restart when a new pattern arrives. In the Adaptation component, Analyze component notifies the Plan component about goal violations using this pattern. *Facade pattern* provides a set of interfaces to a group of components. This is used to provide interfaces for the Preprocessing and Training, and the Plan component. *Command pattern* helps to decouple the caller and receiver of a request. It has been used to validate the feature selection from the Plan facade and pass to the effectors. This helps to separate the effectors and the Plan component.

The logical view indicates effective adaptation and separation of the adaptation logic from the business logic. The structural view enables reuse of the whole subcomponents and their classes. So, the proposed methodology supports effective adaptation with component and subcomponent-level reuse.

III. CASE STUDY: ZNN.COM

Znn.com is a model problem used in numerous papers [7]. It is a news serving application where a load balancer is connected to a server group. Its business goal is to serve with a minimum content fidelity and within the budget while maintaining a minimum performance. These interrelated goals demand a self-adaptive mechanism to operate optimally.

In Znn.com, every server is an application feature as these need to be added or removed at runtime. Content fidelity types (high, low and text) are application features as these can be toggled. Server and content fidelity features belong to *at-least-one-of* and *exactly-one-of* dependency types respectively. Performance, content fidelity and cost can be calculated by response time, content size and number of active servers respectively. Service time and request arrival rates can be considered as environment features.

A. Experimental Setup

Znn.com was deployed on five virtual machines running Apache2 web server, which were connected to a load balancer. Two more virtual machines were used to collect metric values and to simulate user requests respectively. An adaptation component was developed in Java following the proposed approach and incorporated with Znn.com.

Prior to the experiment, the inputs mentioned previously were provided. Moreover, in a simulated environment, Queueing Theory was used to calculate response time where the M/M/c queue model was utilized to represent a system with c servers. Reusability was evaluated using metrics mentioned in Table II. To assess effectiveness, an experiment similar to [8] was performed with a higher load, which is, 1) 15 seconds of load with 30 visits/min 2) 2.5 minutes of ramping up to 3000 visits/min 3) 4.5 minutes of fixed load to 3000 visits/min 4) 9 minutes of ramping down to 60 visits/min.

This experiment was performed five times starting from a single server and high fidelity feature selection as this results in the worst performance. The load was increased by 120 visits/min on every run and the system reached its maximum memory limit after five runs. Following the literature, the main objective (response time) was compared in two situations, namely adaptation and without adaptation [2][7][9].

TABLE III. CLASS-LEVEL REUSABILITY METRIC VALUES

Metric	Components	Mean	Max	Min	%-Acceptable Classes
LCOM4	Monitor	1	1	1	100
	Analyze	1	1	1	100
	Plan	1	4	1	87.5
	Execute	1	2	1	75
	Learning	1	2	1	80
	KBC	1	2	1	85.7
DIT	Monitor	1.33	2	1	100
	Analyze	1	1	1	100
	Plan	1.35	2	1	100
	Execute	1.25	2	1	100
	Learning	1.2	2	1	100
	KBC	1	1	1	100
AC	Monitor	1.14	2	1	87.5
	Analyze	1	1	1	100
	Plan	1.37	5	1	84.21
	Execute	1.33	3	1	83.33
	Learning	1.13	2	1	87.5
	KBC	1	1	1	100
WMC	Monitor	2.67	8	1	100
	Analyze	2	3	1	100
	Plan	4.6	18	1	100
	Execute	5.5	15	1	100
	Learning	2.92	9	1	100
	KBC	2.83	9	1	100

TABLE IV. COMPONENT-LEVEL REUSABILITY METRICS VALUES

Metric	Monitor	Analyze	Plan	Execute	Learning	KBC
RCO	0.17	0.33	0.29	0.33	0.25	0.2
RCC	0.33	0.33	0.29	0.33	0.5	0.6
SCCr	1	1	1	1	0.67	1
SCCp	1	0.67	0.67	1	0.67	0.75

B. Metrics

Table II shows the metrics used to assess reusability of Monitor, Analyze, Plan, Execute, Learning and KBC components. Reusability was evaluated for the whole component as well as for its classes. Four popular reusability metrics by Washizaki et al. were used to evaluate component-level reusability [10] (Table II). Reusability of the classes was evaluated by Lack of Cohesion of Methods 4 (LCOM4) and Afferent Coupling (AC) as these are well-known and valid reusability metrics [11][12]. Depth of Inheritance Tree (DIT) and Weighted Methods per Class (WMC) were also used as these are well-understood and well-validated [13]. The thresholds for LCOM4, WMC, DIT and AC are provided in [11], [14] and [15]. It is notable that multiple metrics have been used as no single metric can represent the overall reusability of the system [12].

1) Reusability of Adaptation: Table III summarizes the reusability metric values for classes from each aforementioned component. It shows the minimum, maximum and average of the metric values for the classes and the percentage of acceptable classes according to the metric thresholds. From the table, the mean LCOM4 values are close to the ideal value (i.e.,1). Here, Execute component has the lowest acceptable classes as it contains system-dependent customizable effectors. For DIT and WMC, all the classes are acceptable as per their

 ISBN: 978-1-61208-590-6

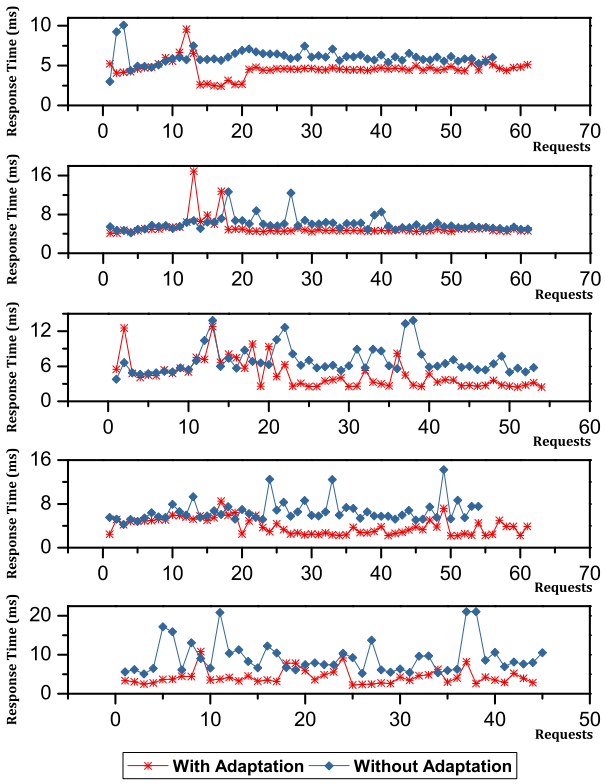

Figure 4. Comparison of Performance: Adaptation vs Without Adaptation.

thresholds. However, the average WMC values are much lower than the threshold because using design patterns have resulted in smaller methods. For AC, 87.5, 100, 84.21, 83.33, 87.5 and 100 percent classes are acceptable. Here, Execute has the lowest AC value for the aforementioned reason.

The component-level reusability metric values are shown in Table IV. Here, all the Rate of Component Observability (RCO) and Self-Completeness of Components Return Value (SCCr) values are within the threshold. The RCC values are also within the acceptable range except for Learning. This is because Learning component is highly customizable as all the preprocessing, learning algorithms etc. can be easily substituted. For Self-Completeness of Components Parameter (SCCp), only Monitor and Execute have out of range values as these depend on the metric API and system-specific effectors respectively.

2) Effectiveness of Adaptation: Figure 4 shows the five runs of the experiment. By analyzing the system average performance, the response time threshold was chosen to be 6.2 ms. In the first run, response time gradually decreases after about 12 requests and rises after about 20 requests. Then, the response time is almost constant due to the constant load scenario (Section III-A). With adaptation, the response time gradually decreases under 6.2 ms and remains as such. However, without adaptation, response time remains more frequently over 6.2 ms. Second run shows a similar pattern.

In the third run, for with adaptation scenario, the response time gradually drops down the threshold after about 22 requests and remains stable up to about 36th request when a sudden performance goal violation occurs. However, adaptation quickly reduces the response time under the threshold. The fourth run shows a similar structure. The fifth run represents the highest load run of all. In this case, the system becomes unstable and response time varies a lot. However, the mechanism without adaptation produces response time above the threshold where the system with adaptation crosses the threshold only about five times, but runs down within threshold instantly.

TABLE V. COMPARISON OF REGRESSION MODEL ACCURACY WITH AND WITHOUT ENVIRONMENT FEATURES

Runs	With Environment Features			Without Environment Features		
	RMSE	Adjusted R^2	Correlation Coefficient	RMSE	Adjusted R^2	Correlation Coefficient
1	0.7428	0.6637	0.7093	0.9373	0.278	0.4553
2	0.8626	0.7804	0.8683	1.6502	0.12322	0.3184
3	0.862	0.787	0.869	1.6444	0.16069	0.3439
4	0.7944	0.8114	0.888	1.4944	0.28563	0.5043
5	0.7884	0.8126	0.8946	1.6054	0.19748	0.4165

Table V shows the accuracy of the regression model regarding environment features. In this case, three metrics, namely Root Mean Squared Error (RMSE), Adjusted R^2 and Correlation Coefficient are used. Among these, RMSE is smaller by 0.6563 on average considering environment features. Adjusted R^2 and Correlation Coefficient are higher by 0.562 and 0.4382 on average respectively. These indicate that considering environment features results in more accurate metric prediction, and so, better adaptation decision.

C. Discussion

The following observations can be made from the results.

- The class reusability metrics indicate overall high reusability of the component classes on average. The 100 percent accepted classes for DIT and WMC, and low mean WMC values indicate that using design patterns have resulted in classes with smaller methods and lower inheritance depth. This makes the behavior of the classes more predictable. Besides, LCOM4 and Afferent coupling indicate that an overall higher cohesion and lower coupling is achieved, leading to higher reusability.
- The component reusability metrics indicate that all the components have higher reusability. However, Learning has the highest customizability and, Monitor and Execute have external dependencies.
- Figure 4 and Table V indicate that adaptation improves the performance of the system gradually. As knowledge base is gradually enriched, this justifies the effectiveness of this process. Moreover, the accuracy measures for the first run from Table infers that knowledge base generation provides useful training data. The high accuracy scores also indicate that adaptation decisions are effective.
- Table V infers that accuracy largely suffers when environment features are not considered. This validates the use of environment features in the proposed approach.

IV. RELATED WORK

In the literature, most of the learning-based self-adaptive systems aim to achieve effectiveness. Kim et al. proposed a Q-learning-based approach where learning derived Q-values and adaptation actions with maximum Q-values were chosen [9]. Han et al. proposed a reinforcement learning-based approach where learning discovered the model of the environment in context in order to pick adaptation policies [16]. None of [9] and [16] considered application factors and reusability. Elkhodary et al. proposed supervised learning-based FUSION

technique, which applied learning to derive relationships between application features and metrics, and used these to optimally select features [2]. However, environment features were not considered. FUSION tool also could not be effectively reused as it needed to be changed from system to system [2]. It also could not be applied when training data was not available.

A few techniques that address reusability have been proposed. Garlan et al. proposed the Rainbow framework [1] for adaptation with infrastructural reusability. Rainbow captured commonalities using architectural styles where systems with same architectural style could reuse elements such as rules, parameters etc. However, reuse among different architectural styles was limited. Component Model-based approaches such as the K-Component Framework [17] and the Fractal component model-based approach [18] relied on structuring the system with a specific component model for reusability. However, a specific component model made reuse between different component models costly because the full code base needs to be refactored. Ramirez et al. produced a list of twelve design patterns for self-adaptive systems and applied these in Rainbow [1]. However, it would be better if these patterns could be mapped into more well-known GoF patterns [4].

None of the proposed techniques consider application and environment features together for more effectiveness. Besides, the learning-based approaches do not consider increasing reusability. Learning-based approaches like FUSION also fails if training data is absent. The proposed approach overcomes all these by considering application and environment features, applying design patterns for reusability and providing a training data generation mechanism.

V. Conclusion and Future Work

This paper introduces an adaptation component design considering reusability and effectiveness. A knowledge base constructor is presented that randomly toggle features and considers corresponding metric values to derive training data. This data is used to produce equations to predict metrics from application and environment features using Machine Learning. These equations and the feature dependencies help to derive an optimization problem. Solving this, a feature selection with maximum total utility function value is obtained, which can be executed through customizable effectors. This overall generic logic supports component-level reuse. Design patterns are used to enable reuse of the subcomponents. The reusability metric values for each subcomponent are within the acceptable threshold, indicating high reusability. Adaptation effectiveness is also achieved as the system gradually decreases the response time under a provided threshold when goal violation occurs. The learning accuracy regarding environment features also validates adaptation effectiveness and utilization of these features.

In future, the technique will be enhanced to take adaptation decision by foreseeing future effects of the decision on the system. It will also be extended to automate threshold selection for metrics and failed adaptations.

References

[1] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *IEEE Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[2] N. Esfahani, A. Elkhodary, and S. Malek, "A learning-based framework for engineering feature-oriented self-adaptive software systems," *Software Engineering, IEEE Transactions on*, vol. 39, no. 11, pp. 1467–1493, 2013.

[3] A. J. Ramirez and B. H. Cheng, "Design patterns for developing dynamically adaptive systems," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2010, pp. 49–58.

[4] M. L. Berkane, L. Seinturier, and M. Boufaida, "Using variability modelling and design patterns for self-adaptive system engineering: application to smart-home," *International Journal of Web Engineering and Technology*, vol. 10, no. 1, pp. 65–93, 2015.

[5] IBM Corporation, "An architectural blueprint for autonomic computing," *IBM White Paper*, 2006.

[6] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[7] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," in *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*. ACM, 2006, pp. 2–8.

[8] S.-W. Cheng, *Rainbow: cost-effective software architecture-based self-adaptation*. ProQuest, 2008.

[9] D. Kim and S. Park, "Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software," in *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*. IEEE, 2009, pp. 76–85.

[10] H. Washizaki, H. Yamamoto, and Y. Fukazawa, "A metrics suite for measuring reusability of software components," in *Software Metrics Symposium, 2003. Proceedings. Ninth International*. IEEE, 2003, pp. 211–223.

[11] M. Hitz and B. Montazeri, "Measuring coupling and cohesion in object-oriented systems," in *Proceedings of International Symposium on Applied Corporate Computing*, 1995, pp. 25–27.

[12] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[13] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[14] R. Shatnawi, W. Li, J. Swain, and T. Newman, "Finding software metrics threshold values using roc curves," *Journal of software maintenance and evolution: Research and practice*, vol. 22, no. 1, pp. 1–16, 2010.

[15] K. A. Ferreira, M. A. Bigonha, R. S. Bigonha, L. F. Mendes, and H. C. Almeida, "Identifying thresholds for object-oriented software metrics," *Journal of Systems and Software*, vol. 85, no. 2, pp. 244–257, 2012.

[16] H. N. Ho and E. Lee, "Model-based reinforcement learning approach for planning in self-adaptive software system," in *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*. ACM, 2015, p. 103.

[17] J. Dowling and V. Cahill, "The k-component architecture meta-model for self-adaptive software," in *International Conference on Metalevel Architectures and Reflection*. Springer, 2001, pp. 81–88.

[18] P.-C. David and T. Ledoux, "Towards a framework for self-adaptive component-based applications," in *Distributed Applications and Interoperable Systems*. Springer, 2003, pp. 1–14.

 ISBN: 978-1-61208-590-6