Architecture-Based Dynamic Reconfiguration of Composite Components on Model-Based Reinforcement Learning

By: Yonas Moges

A Thesis Submitted to the Department of Computing

School of Electrical Engineering and Computing

Presented in Partial Fulfillment for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

February, 2019

Adama, Ethiopia

Architecture-Based Dynamic Reconfiguration of Composite Components on Model-Based Reinforcement Learning

By: Yonas Moges

Name of Advisor: Mesfin Abebe (PhD.)

A Thesis Submitted to the Department of Computing

School of Electrical Engineering and Computing

Presented in Partial Fulfillment for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

February, 2019

Adama, Ethiopia

Declaration

I hereby declare that this MSc thesis is my original work and has not been presented as a partial degree requirement for a degree in any other university, and that all sources of materials used for the thesis have been full acknowledged.

Name: Yonas Moges

Signature: _____

This thesis has been submitted for examination with my approval as thesis advisor.

Name: Mesfin Abebe (PhD.)

Signature: _____

Date of Submission: _____

Acknowledgment

Foremost, I would like to thank the Ultimate God for his almighty, and his mother Saint Merry. I am thankful Lord for everything that you allow to cross my path. I am also thankful for the decisions that you allowed me to make and the lessons that come from these decisions.

I respect and thank Mesfin Abebe (PhD.), for providing me an opportunity to do the research work in dynamic adaptive systems and giving me all support and guidance which made me complete the research duty. I am extremely thankful to him for providing such nice support and guidance, although he had a busy schedule managing the coordinating affairs.

I would like to express my deep sense of gratitude to Mr. Kibrom Tadesse. I am thankful and fortunate enough to get constant encouragement as well as your continuous initiatives as a big brother starting from BSc project work.

My sincere thanks also go to all my family: Abrish, Tsehaye, Hailat, Teme, Ame, Efi and the little one Baby, if you were not all that, I would not have been able to do this work very easily. Dear Mom, God give you the gift you gave me and for all that you have done to me while you are alive. This thesis is dedicated to my mother, Werkinesh Welde.

The Twins, Hymi & Hiwi, I never forget the help of your assistance, God always watches over you. I hope you will always get close to my movement. Thank you for being the cornerstone of my success to accomplish this work.

I would like to thank all ASTU's CSE department staff members, whose helps me to complete my dream starting from BSC class up to now by motivating and helping me to deliberate the required result. Additionally, I also would like to thank my fellow Smart Software SIG (3S) Lab mates for their appreciation to finish this work.

Finally, it is my radiant sentiment to place on record my best regards, deepest sense of gratitude to my friends Shegaw, Ashenafi, Abebaw, Yidnekachew, Samuel for their timely help and wholehearted support rendered to me in the completion of my thesis.

Table of contents

List of Tables

List of Figures

List of Equation

List of Abbreviation

ABSA	Architecture-Based Self-Adaptation
ActivFORMS	Active FORmal Models for Self-adaptation
AI	Artificial Intelligent
AOP	Aspect-oriented programming
API	Application Programming Interface
CPU	Central Processing Unit
DevOps	Development Operation
DGCL	Dijkstra's Guarded Command Language
DSRACC	Dynamic Software Reconfiguration Architecture in Composite Component
DTMC	Discrete-time Markov Chain
GRAF	Graph-based Runtime Adaptation Framework
IBM	International Business Machine
MAPE-K	Monitor, Analyze, Planning, Execute-Knowledge
MBRL	Model-based Reinforcement Learning
MDP	Markov Decision Process
MFRL	Model-free Reinforcement Learning
MOFS	Model to Text transformation toll
MW	MiddleWare
OL	Ontology Language
OPERA	Optimization, Performance, Evaluation, and Resource Allocator
QA	Quality Attribute
QoS	Quality of Services
RL	Reinforcement Learning
SAFU	Self-Adaptive Fitness Unit
SAS	Self-adaptive System
SL	Supervised Learning
SLR	Systematic Literature Reviews
SQL	Sequential Query Language
TA	Timed Automata
TCTL	Timed Computational Tree Logic

UB	Utility-Based
USL	Unsupervised Learning
WEKA	Waikato Environment for Knowledge Analysis
XTEAM	Xtensible Tool-chain for Evaluation of Architectural Models

Abstract

Software systems are subjects to continuously change and they are also being affected by dynamic environments such as changes in user requirements, priority changes over user requirements, execution resource changes and performance reduction. Therefore, systems should have reconfigured their architectural elements dynamically at runtime to mitigate unpredictable circumstances which are unknown at design time as well as to meet some quality goals.

Architecture-based self-adaptive is a well-known approach to tackle uncertainties at runtime by providing functionalities described in monitor-analyze-plan-execute and knowledge feedback loop to detect the environments, analyze the system constraints, planning new configuration and perform adaptation on the target system. For choosing the best configuration, a decision-making algorithm has vital roles. Among that, utility theory is used in existing studies which considers each quality utility, the impact of qualities on adaptation action and quality prioritization one over the other. However, utility theory has static adaptation behaviors that can't alleviate uncertainties exist at runtime. Consequently, this work proposed new frameworks called Dynamic software reconfiguration architecture in composite components by adding new features on the existing Rainbow self-adaptive framework which used utility theory for decision making purpose. Dynamic software reconfiguration architecture in composite components combines the utility theory principle and Model-based Reinforcement Learning strategy to answer how systems adapt themselves to a new behavior. To show the effectiveness of the proposed framework Information delivery self-adaptive system will be demonstrated that can modifying its runtime behavior according to the operating environment changes.

Finally, the study presents results of proposed solution by performing reconfiguration in different ways. As can be seen, the combination of Model-based reinforcement learning and utility theory would yield better decision-making capability for adaptation process. Besides that, the framework also evaluated on the basis of Self-adaptive fitness value with Rainbow, and it exhibited better adaptation fitness.

Keywords: Self-adaptive systems, Dynamic software reconfiguration, Rainbow, Model-based reinforcement learning, architecture-based self-adaptation.

CHAPTER ONE

1. Introduction

1.1. Overviews

In the real environment, software systems are undergoing in some unpredictable circumstances during runtime[1]. These circumstances may be arising from user requirement changes, changing of operating environment execution and the changes of requirements priority one over the others are the reason for self-adaptation [2]. For these reasons, the self-adaptive system makes an adaptation in order to achieve some specific qualities goal. Self-adaptation in software system is described as the reconfiguration of architectural elements includes its components and connectors based on the functions specified in the MAPE-K reference autonomic control loop model [3]. This loop, MAPE-K provided the solution for architecture-based self-adaptation methodology. The main activities of this control feedback loop also described in MAPE-K reference model which illustrate multiple activities called monitoring, analyzing, planning and executing over shared knowledge [3].

Above all, using machine learning strategies as a decision making [4] will provide a solution to systems that can have an ability to self-reconfiguration contextually that means, a system can choose a new configuration by exploiting its prior experience. The self-adaptive system can also make an analysis of its adaptation impact before taking action during the reconfiguration process because of machine learning strategies. Thus, systems can have the ability to determine strategy selection to increase the reward of its action and learning the future simultaneously.

Sometimes systems can specify their learning algorithms at design time in a programmed fashion, [5] [6]manner which specifies every possible interaction of environments and configuration is mapping by developers. In this strategy, a programmer cannot anticipate all possible mapping of environments state to adaptable software configuration. Consequently, there might be some unforeseen situation i.e. called uncertainty at runtime. In contrary, if the system designed with learning capability at runtime and has an ability to learn from its previous action in the basis of online planning [7] will have a probability to tackle uncertainty during operation time. To implement online-planning learning strategy for adaptable architecture [8], the system should compose an ability to identify the adaptable model, continuously monitoring systems environments,

planning adaptable software architecture, analysis the impact of reconfiguration and executing the strategy activities in its adaptation processes [9].

Based on the description in [10] self-adaptive software engineering roadmaps, there are several issues that should be considered during designing and developing such systems. According to that, a self-adaptive system needs to continuously detect the environments, analyzing the collected information against to the architecture, planning or choosing the new configuration that can be suitable for new environment states and performs the configuration on systems and finally starts detecting environmental runtime metrics repeatedly. On this path, it makes a feedback loop that controls the reconfiguration pattern either in centralized or decentralized control pattern [11], that describes the collaboration and communicating components and connectors during reconfiguration. In centralized reconfiguration pattern, a single component controls all reconfiguration action performs in the whole systems, in other direction, in decentralized reconfiguration pattern every components and connector perform reconfiguration by itself locally.

The organization of the feedback loop can be either single feedback, two or multiple feedback loop as based on the multiple reconfiguration reasons [12]. The obvious things are when the feedback loop increases the complexity of the system will be raised in addition to the nature of complex behaviors of self-adaptive systems. There needs to clearly identify what kinds of resources are targeted for reconfiguration purposes? Systems can modify their internal structure by changing their services, architectural elements, parameters. So that there are specific adaptation actions or strategies e.g. adding components, removing components, that leads the system to change itself dynamically. Following the adapting to a new environment, the identifying reason behinds adaptations are also core steps because it leads to why the system needs to change their runtime behaviors? The answers would different according to the application areas e.g. the robot changes its internal structure, when it wants to change the path or wants to pass the obstacle encountering in front of it, to get maximize quality goals, to provide services and resources according to the distribution for using them effectively, those all needs can be generalized as system needs reconfigure itself because of mitigating the requirement changing and unpredictable circumstances that happening at runtime.

After the identification of feedback loop structures, reconfiguration patterns, targeted resources that being adaptable and the reason behinds adaptation, major issues are followed. This issue is

designing decision-making algorithms makes a decision on whether reconfiguration is needed or not. The decision-making algorithm creates differences on self-adaptive characteristics by classifying into static reconfiguration strategies which is all decision-making strategies are specified at design time by following offline-learning or dynamic reconfiguration that enables self-adaptive systems to reconfigure its structure dynamically at runtime by using online planning. So that, self-adaptive systems can use utility theory-based, machine learning strategies and a set of routines [13]for their decision-making purposes. The utility theory takes in consideration of utility of each adaptation action in relation quality attributes as a utility function, the impact each quality attributes on adaptation strategies and the priority of qualities relative to each other as utility preferences. The framework Rainbow [14], is a well-known framework in which it describes how to design and develop self-adaptive systems on an architecture-based self-adaptation methodology which uses utility-based decision-making algorithms.

The other decision-making algorithms, machine learning kinds, supervised learning and unsupervised learning approaches have better abilities for prediction and adaptation than programmed fashion approaches [15]. However, these approaches need trained data and set of examples or labels for their learning strategies which are the difficult part for mapping of situations to the corresponding reconfiguration action. Reinforcement learning can mitigate the difficulties of preparing a set of examples by using feedbacks of the agent after the agent performs a specific action on current states in the environment by accumulating rewards. So using reinforcement learning for developing self-adaptive is better than using other approaches of machine learning [16].

One broad class of RL is model-based reinforcement learning, which can learn from the environment and concurrently predict the impact of the action on the model. Due to this behavior, the decision-making of self-adaptive systems can be constructed with a model-based reinforcement learning approach which manages uncertainty and performs model verification by quantifying the current architecture to the current environment on the model. On the contrary, the model-free RL also makes only learning without predication of the next state and action, but it makes a decision tree more complex [14]. On both of classes how state and action are choosing or planning from numbers of learning situations, means from accumulated experiences in the model that leads to exploitation and exploration strategies [9]. The assurance and validation processes, at the final stage, should be considered in the development of self-adaptive systems is the way of checking the newly

reconfigured system keeps consistently with its functional and non-functional constraints. For that matter, the adapting and the adapted module must consider the separation of concern concept to perform reconfiguration safely.

Based on the general overviews of how self-adaptive systems design and developments, this research study proposed a new framework by adopting the existing Rainbow framework with additional components for decision-making purpose using architecture-based self-adaptation methodology. The utility based and model-based reinforcement learning strategy for planning and choosing a new configuration is includes into the proposed framework. The proposed framework provides clear environmental states and actions information that encompasses to knowledge-base model to learning algorithms to maximize accumulated rewards (goals) in decision-making steps. Besides that, the framework can also manage uncertainty that happens at runtime compared to MFRL and UB decision-making. By applying the working principles of the framework in the developing prototype, it is possible to exhibit a better performance with respect execution time. For reason out, the results will be conducted with utility-based adaptation, model-free RL, model-based RL as well as by combining UB with model-free RL and model-based RL adaptation.

1.2. Motivation

The researchers have noticed frequently the changing of executing environment and user requirements in a software development world affects the continuous usage of software and reusability of its elements. Therefore, rebuild and reconfiguration of the software components is necessary to tackle this dynamicity by the involvement of human. However, this action is very tedious and expensive in different cost such as time, budget, human power, as well as in computing resources like memory. As a fact, software architecture is the backbone of software design and development, since every organization, structure and internal communication among components are specified and directed by the architecture. As a solution, we thought dynamic reconfiguration of software components and connectors at runtime will solve the problem seriously, indirectly such action will change the architecture's topology of software to another. Besides that, if the architecture of software has learning capability through machine learning strategy, it can reconfigure its element at runtime dynamically without the involvement of human. The authors read some papers in the area of dynamic software architecture reconfiguration at runtime. Most old papers had described the reconfiguration process in programmed fashion, which is specified at design time and implemented

during construction by the programmer. Others research also solved the problem of machine learning based software architecture reconfiguration dynamically in a self-adaptation manner and proposed different methodologies in robotics activity, smart home application, networking architecture, image reconfiguration and web security implementation on cloud resource allocation. Based on their different implication we have motivated to propose a framework that provides environments to designs and develop a simple pc-based application that performs reconfiguration dynamically on its architectural components which have aggregation and composition-relationship among them. In this research, we wish to perform reconfiguration activities only using attach and detach operations on composite component's connections and adding and removing components at an operational time based on utility theory-based and model-based reinforcement learning strategies for decision making to meet some quality attributes by adding extra features on Rainbow frameworks with the managed prototype.

1.3. Statement of the Problem

Dynamic reconfiguration with static planning can't foresee the uncertainties happening at runtime and didn't solve the user requirement changes on the quality of the software which uses utility outcomes for the decision-making process. Reconfiguration scenarios also implemented as programmed fashion. As a result, a programmer should have to completely map all requirements change to a new configuration. Such mapping might be encoded set of rules for 1000 state and configuration which tedious duty. But when 1001^{st} state encounters at runtime, there will be a failure of service because not rule is encoded for the newly emerged state [17].

Model-based reinforcement learning exhibits slow performances [9] due to the additional activity of predicting and checking the model besides learning activity. As a result, it will degrade the performance of the self-adaptive system. Consequently, that decreasing the state-action selection spaces and model-checking time are major important to enhance better performances. Beside that managing the exploitation-exploration tradeoff in reinforcement learning is would depended on the given epsilon the needed accurately model the environment which is impossible.

Self-adaptive systems by nature have a complex internal structure and make loops from continuously triggers one to other components [18]. Therefore, if the system is organized under composite components which maximize the level of internal structure complexity, it would be

considered as uncertainty [19] at runtime. So, it is crucial work to over tackle such uncertainty because of in the current world systems have high internal structures during their development runtime.

Based on the above gaps/problems, the following are some questions in the proposed research that will be addressed thoroughly:

- What result exhibited from the combination of utility theory adaptation and model-based reinforcement learning strategy? Is possible in architecture-based software reconfiguration?
- What advantage would gain from the implementation of reinforcement learning strategy according to the MAPE-K autonomous feedback loop in all kinds of reconfiguration ways?
- How the uncertainty raised from the complexity of internal structures could be mitigated in the proposed work clearly?

1.4. The Objective of the Study

1.4.1. General Objective

The main objective of the research study is designing and developing architecture based dynamic reconfiguration for composite components using model-based reinforcement learning.

1.4.2. Specific Objectives

- Aligning the MAPE-K components and MDP environments to architecture-based self-adaptation methodology with reinforcement learning problems.
- Representing the reinforcement learning problems into a self-reconfiguring software system.
- Constructing an architectural model and environmental model that can accommodate relevant information or knowledge.
- Implementing feedback loop structures to controlling purposes for the prototype to evaluates the result of the study.
- Discussing the evaluation of multiple adaptation ways and the proposed framework by comparing with the Rainbow existing framework on the basis of DAFU

1.5. Scope and Limitation

1.5.1. Scope

The reconfiguration of architectural topology is performed by creating and removing components and connectors. Due to a head tight schedule operation like passivate and activate of architectural elements will not be issued.

The prototype is developed using only a single feedback control loop from types of feedback loop application styles.

1.5.2. Limitation

During designing and developing of the self-adaptive system some observations and approaches are needed from industries [20] that help to elaborate the research roadmap in further. Unfortunately, there is a lack of local companies and industries which develop systems based on dynamic software architecture style. As a result, the raw data for the proposed research is going to be extracted from the literature reviews. Thus, the impact of additional data and current experimental environment will change the outcome of the result.

The framework didn't include Adaptation failure strategies, due to the limitation of time and other necessary criteria to focus on it [21].

1.6. Research Methodology

The proposed methodology implements a new adoption framework from Rainbow that enables to build self-adaptive system which has composite components to accomplish the specific objective and to fill the gap stated above. For that matter, the study uses architecture-based self-adaptation methodology. That is because; it states the necessary actions and requirements and inevitable steps where self-adaptive systems should be satisfied. Additionally, a research roadmap of software engineering of the self-adaptive system [10] is used for justification and evaluation of the development structures of the proposed prototypes. In general, the methodologies are composed as follows.

***Problem identification*:** on this phase problems are identified and justified.

***Preparing the objective of a solution*:** by refining different books, important articles from literature and identifying necessary procedures, techniques, and requirements.

***Design and development*:** by adding new features on existing frameworks according to ABSA guidelines the follow sub-methods are used.

- Method to feedback loop arrangement – single feedback loop in model-based learning fashion.
- Method handles reconfiguration process – combining utility-based adaptation with the model-based reinforcement learning problems.
- Method to handle multiple configuration trade-off – Utility function for quantification of the current architecture to the current environmental state.
- Method to make stochastic behavior analysis - Markov chain on the basis of Markov decision process that considers state, new state, action, transition probability, and rewards.
- Method for controlling strategy in feedback loop – Centralized control pattern.
- Method for validation and evaluation for assurance - Testing and Model verification

***Implementation*:** utility function, the impact of adaptation strategies of each quality of the system and utility preferences of values used with the Markov decision process to represent the environment of RL problems will be demonstrated. The test code developed using IntelliJ IDEA editors while java used for user interface and MAPE-K feedback loop implementation code and python [22] used for coding the reinforcement learning problems of the prototype.

***Evaluation*:** results from the demonstration will be analyzed to validate the consistency of the prototype's architectural constraints after reconfiguration using model verification and stochastic behavior analysis by DTMC. Besides that, the whole proposed solution will be evaluated against registered performance execution. The discussion is drawn from the perspective of the existing literature's hypothesis.

1.7. Beneficiary and Application of Results

The main beneficiaries of the proposed study are any stakeholder that using self-adaptive software systems. On the perspective of users, the functionality of the system would be confronted when unpredictable circumstances encountered at operation time i.e. the user can manage its operating environments efficiently and protects the additional costs for maintaining the systems due to the arising of uncertainties and requirement changes on the side of the qualities of the system.

On the corresponding of any software developers, it minimizes the developing cost, efforts and time as maintain, and developing new software as the operating environment changes. It generates better revenue if the systems have learning capabilities in dynamic environments. The researchers also the other beneficiary because of they can get a chance to illustrate the proposed works for other extending their research work, authors also can develop how to analyze research concepts, learns the way of finding a new solution scientifically helps to maximize the experiences related to research

1.8. Thesis Organization

The research paper is organized under 7 chapters, in which *chapter one* is discussed about the introduction of the study, the motivation, and the statement of problems, the research questions that would be answered in the proposed solutions, the scope and limitation, the objective of the study, the methodology and the beneficiaries of the application results.

Chapter two is discussed about the background literature on self-adaptive systems, architecture-based self-adaptation methodology, MAPE-K autonomic computing loop with its constitutes, Rainbow framework, with the application called Znn.com, about dynamic software architecture, machine learning and categorization of machine learning algorithms with RL, composite components. Additionally, the chapter also discussed the characteristics of self-adaptive systems, how to introduce learning into self-adaptive systems and how are the decision making and tradeoff analysis of multiple QAs performed at runtime during reconfiguration? Questions are raised. Finally, it discussed on reviews result and related work.

Chapter three discussed the ABSA methodology and methods that are encompassed under different phases of the study was performed, it starts discussing from literature collection to evaluation and

discussion phases. What data type is serves as input for the research, model verification strategies at runtime, and the evaluation metrics that done at result discussion chapters.

Chapter four, which is the proposed solution, discussed the adopted proposed frameworks with its internal constitutes how to represent the self-adaptive environments, the alignment of decision dimension of self-adaptive systems with the MAPE-K autonomic loop and MDP. The algorithms of utility-based adaptation, and reinforcement learning problems with the combination of both which is proposed algorithm, model construction, and finally discussed prototype demonstration in the framework.

Chapter five which is Implementation Discussed about, the working environments of the case study, the adaptation inputs, high level architectures of the prototype, sample source code of components that feasible both the managed and managing modules in the framework, the utility-based adaptation with reinforcement learning adaptation, and discussed on the code level of learning strategies.

Chapter six discussed about the prototype's results based on the collected runtime metrics, discussion on the actual results of adaptation by comparing it with the predicated set of state and action, in UB, MFRL, MBRL, UB+MFRL and UB+MBRL adaptation results and finally evaluate the whole proposed methodology against the performance of each adaptation ways by combining model verification for assurance and validation the self-adaptive systems.

Finally, *Chapter Seven* is discussed about the conclusion, recommendation and the future works that recommend by the author.

CHAPTER TWO

2. Literature Review

This part provides a brief background about the domain of architecture-based dynamic reconfiguration of components using machine learning, the criteria to consider in self-adaptive systems, the requirements needed to design and implements self-adaptive systems concepts which are collected from some preliminary works in the systematic literature review and related area works under similar methodology.

2.1. Overview

Architecture-based self-adaptation has emerged as a method in 2004 with the proposing of autonomic computing loop which is called MAPE-K by IBM. After that; Rainbow framework [23] has developed for architecture-based software reconfiguration systems by using MAPE-K feedback loops and developing a managed system that can be adapt using it. This method had risen to tackle dynamicity environments that lead to the existence of uncertainty. Then after, a number of types of research have been proposed solutions on dynamic architecture and self-adaptive systems [24]–[27]. But much of the existing literature on self-adaptive systems is focused on programmed fashion reconfiguration style and reconfiguration based on aspect-oriented software developments paradigm. Besides that; some of the literatures were proposed the dynamic software architecture reconfiguration through some intended entities. The intended entities might be parameter changes [24] or component configuration changes [18] which specified the target resources are being a victim from adaptation. The work Costa-Soria, Cristóbal, Jennifer Pérez, and Jose Ángel Carsí, 2009, performs dynamic reconfiguration on composite components, which are composing other components internally that need additional reconfiguration capability when components are reconfigured.

In the next section, some major concepts and works that are written in the architecture-based self-adaptation method will be discussed briefly.

2.2. Self-Adaptive System

In a short description, SAS is a system that can change and evolve their runtime behavior in order to meet some quality goals [19]. Among different meanings of the self-adaptive system from numerous reviewed results; the following two definitions are close to this research work.

Definition 1**:** *Self-adaptively is the capability of the system to adjust its behavior in response to the environment. The "self" prefix indicates that the systems autonomously decide (i.e., with minimal or no interference) how to adapt or to organize themselves so that they can accommodate changes in their contexts and environments. While some self-adaptive systems may be able to function without any human intervention, guidance in the form of higher-level objectives (e.g., through policies) is useful and realizable in many systems (Brun et al., 2009).*

In short, definition 1 discussed why the self-adaptive systems achieve adaptation with broad explanation of how reconfiguration is taking in self-adaptive systems. Additionally, definition 2 also explains as:

Definition 2**:** *A self-adaptive system evaluates its own behavior and changes its own performance when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible (Salehie & Tahvildari, 2011).*

In addition to the above definition which focuses on only why systems adapt themselves with the behavior of them, other definitions have discussed with the answers of how systems adapt themselves to the new behavior. Salehie and Tahvildari (2011) decomposed the adaptation mechanism into several methods: monitoring software units (self-awareness) and the environment, analyzing substantial variations, planning how to respond, and executing so that the choices to take effect. This explanation has described the MAPE-K reference model that be discussed in the next.

In most of the existing solutions, the adaptation processes are allocated to an external adaptation manager that is isolated from the application logic. An adaptation manager realizes the four above-mentioned methods, to control the activities of adaptable software. The application logic is programmed in adaptable software that receives signals from the sensors and effectors required for self-adaptively.

2.3. Architecture-Based Self-Adaptation

It is one of well-known methodology to tackle uncertainties in self-adaptive systems. Because of architecture-based methods, it provides a solution by using architectural elements and model objects which represents the architecture of the system and its environments (operating environments and itself). The main functionality of this method discussed in the MAPE-K reference model [28], [29], which is the autonomic model serves as a reference model to build and design a self-adaptive system using feedback and control loop as shown in the figure 2.1 taken from [17]. This well-known method has the ability to adapt some components and connectors at runtime without having to shut down and restart the whole software system. The model is described in the below diagram.

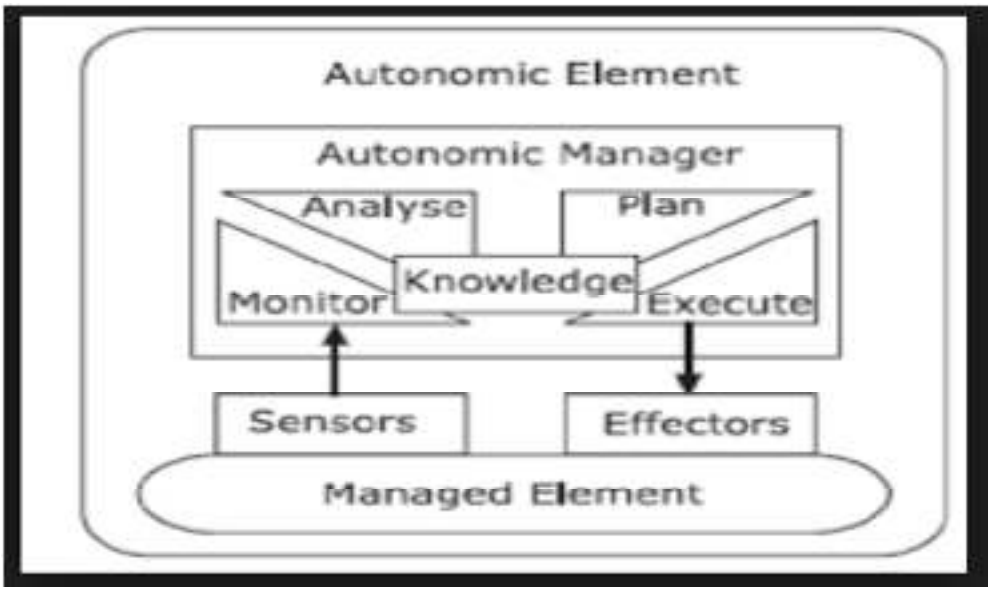


Figure 2.1 Autonomic computing MAPE-K feedback loop

Figure 2.1 shows that Autonomic manager module manages the circumstances of the managed system by using runtime architecture model and environment model. Architecture model is an abstraction of managed systems of architectural elements including components and connectors. Environment model also abstracts the operating resources and constraints as well as the states and actions in which the managed system existing mood. Monitor components in autonomic manager monitor and gathers information about managed systems that trigger the system in adaptation action, then triggers the next component, Analyze. Analyze makes an analysis based on existing resources, checks architectural rule and regulation and evaluates the impact of reconfiguration on the function

of systems and with conformance of architectural constraints, then it triggers and confirms the result to planning component in the module[2] . Then choosing the appropriate configuration to the new environments form prior experiences by learning algorithm would be next action and triggers to the last component which in-turn the managing component execute the new adaptable architecture configuration and make the managed system reconfigure to other architectural topology.

From definition 1, Self-adaptive systems have the capability of modifying their runtime behavior. This modification of behavior at runtime is also the main goal in architecture-based self-adaptive systems. The well-known framework for designing and developing ABSA is Rainbow [23]. It uses systems' architecture as an external controller to manage the target system at runtime. It also serves as basic foundations for many works [1], [3], [26], [30] in the self-adaptation systems area. The framework and its elements are shown in figure 2.2 as [2].

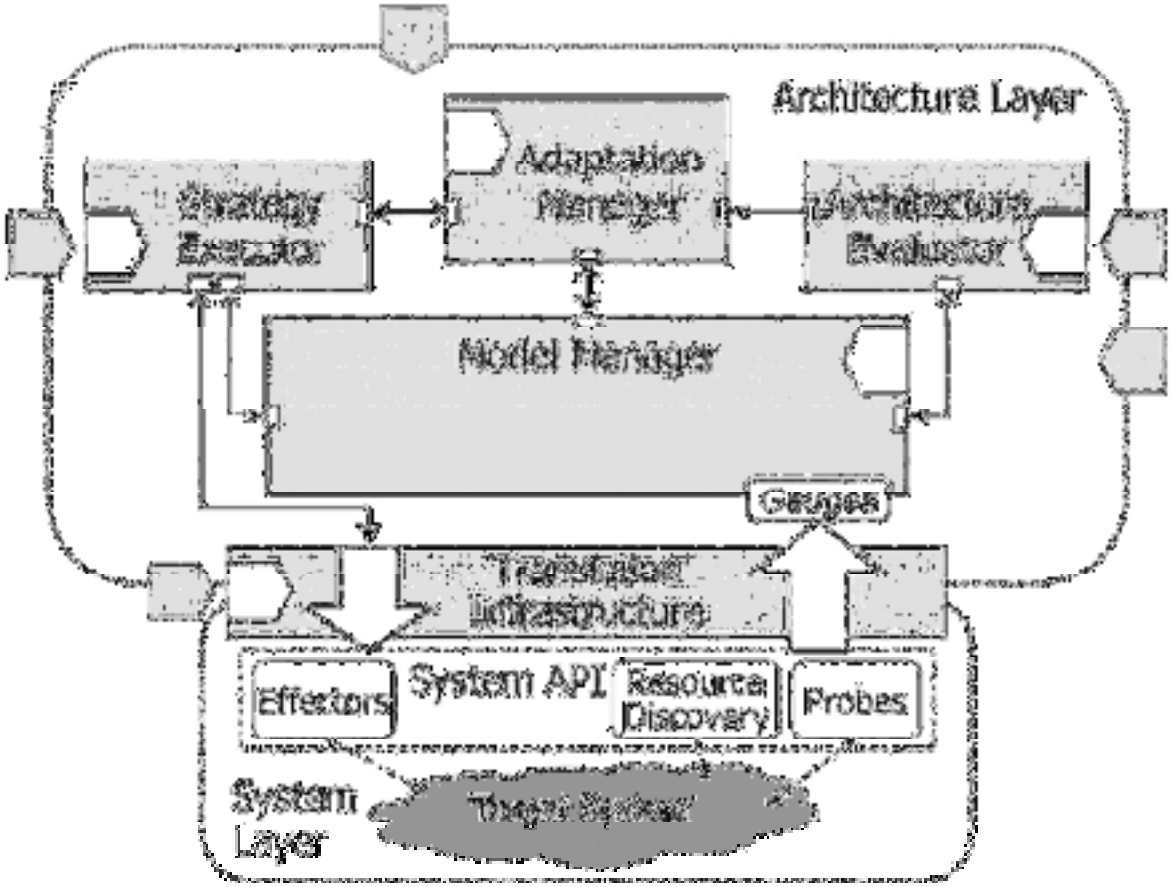


Figure 2.2 Existing Rainbow framework

The framework consist of the following architectural elements

***Model Manager**:* this part includes the knowledge repository and monitoring components in a corresponding MAPE-K feedback loop. It controls both the architectural model and environment model. It also maintains references among the two models through translation infrastructures called gauges. As monitoring components, the model manager continuously detects managed systems by the shared model through accessed API and queries that maintain the equivalent information in probes. As a knowledge repository, it includes architectural constraints, quality goal specification, models state which is determined at design time. Finally, it notifies the Architecture evaluator element of the framework.

Architecture Evaluator: when the model manager detects model changes that need to adaptation it notifies to architecture Evaluator. Architecture Evaluator checks and evaluates the conformance of functional requirements and reconfiguration strategies based on the predefined set of architectural constraints. Rainbow considered the evaluator as rule of architectural topology and behavioral constraints. After it checks violation in a managed system, the Adaptation manager part would notify to trigger adaptation process. This part of the framework also maps to the corresponding Analysis part of MAPE-K components.

***Adaptation Manager*:** Once a violation is detected, there needs a mechanism to decide on the suitable adaptation preparation. When a notification comes from the Architecture Evaluator, the Adaptation Manager procedures the architecture model to select strategy (choose new architecture configuration) that best suits to the existing problem state of the system then complements the execution of that policy. Automating system adaptation involves reinforcing three kinds of information to initiate the process: for what to adapt when to adapt, and how to adapt the system [2]. The planning phase of MAPE-K feedback loops is mapped to Adaptation Manager of Rainbow element. For that matter, the Adaptation Manager determines adaptation strategies, quality dimensions, utility preferences, and the cost-benefit attributes to enable decision making during prioritizing of multiple reconfiguration tradeoffs in self-adaptation action.

***Strategy Executor*:** Once the adaptation strategy is chosen, a mechanism that can transfer the adaptation to the target system would be done. Therefore, the Strategy Executor is triggered by the Adaptation Manager to do this. As well, it resolves model circumstances within the strategy alongside the Rainbow model observes model states and assessments branch conditions to control operators to implement equivalent system-level effectors to transfer changes. It has effectors

components which can perform adaptation action on the managed system. Strategy executor is also analogous components in the MAPE-K feedback loop.

As the researchers' analyzation, the rainbow framework provided solutions for Znn.com application like Cnn.com site which provide news service to their customer. This application consists of multiple servers, clients and load balancer. The server provides news to the customers either in textual mode or graphical mode service quality within specified response time that are set as low, medium and high values which determines budgets that be within or over.

To select the best strategies for adaptation, Rainbow specified utility preferences and quality dimension as in Table 2.1;

Table 2.1 Strategies example in self-adaptive systems

Label	Description	Architectural property	Utility function	Weight
u_R	Avg Response Time	ClientT.experRespTime	⟨(low, 1), (med, 0.5), (high, 0)⟩	0.4
u_F	Avg Content Quality	ServerT.fidelity	⟨(textual, 0), (multimedia, 1)⟩	0.2
u_C	Avg Budget	ServerT.cost	⟨(within, 1), (over, 0)⟩	0.3
u_D	Disruption	ServerT.rejectedRequests	⟨(1, 1), (2, 0.75), (3, 0.5), (4, 0.25), (5, 0)⟩	0.1

The table shows discrete values of each quality dimension that supports to compute the impact of adaptation plans which are a list in customization content highlights of adaptation manager Rainbow component.

2.4. Decision Space Dimensions

Decision space dimensions defined a set of major dimension about the conceivable choices that a developer taking in attention during self-adaptive systems development. The four decision dimensions that are outlined in [10] and processes in learning and adaptive are:

2.4.1. Representation of the Environment

Represent the system at runtime and issues on decision problem exist at runtime is a crucial step in decision space dimension. Therefore, an explicit representation of the internal environment is compulsory to get the observed problems at runtime. This decision dimension cluster has a similar

design like representation in online learning/evolution process[31] , in which represent the environments based on Goal-Scenario state space. The goal is a specific objective that a self-adaptive system should achieve, and Scenario is a collection of some actions to realize the specific objective which means quality goal. This process helps to produce state and action space of the system's environments that help to construct the model of the self-adaptive systems.

2.4.2. Observation of the Self-Adaptive System

This decision dimension describes in detail the role of monitoring components in the MAPE-K loop. Observation decides what kinds of information are detected and when such detections are made for analyzing and choosing a new configuration. Besides that, according to [10], measurement of uncertainties happened after reconfiguration also part of the decision in observation cluster. The two basic questions raised in this cluster are 1) Systems continuously observe the environment and internal representation or some external entity triggers the observation such as timer or external events. 2) What triggers observation and what triggers adaptation and how do these communicate with each other? Continuously detecting method will create complexity and making the system will be busy [32]. For that reason, it better uses external timers to overcoming the problems.

2.4.3. Control

The main apprehension of this decision is computing concrete groups of possible organizations, behaviors, states, parameter values, etc.

It will be represented as calculation of rewards based on MDP for the impact of model analysis. In addition to that identification of systems resource and execution environments also helps to identify whether reconfiguration is needed or not. This design decision wills directly mapping to the analyzing component of the MAPE-K loop.

2.4.4. Adaptation Mechanism

This design decision describes the mechanisms in which a system chooses a new architectural configuration at runtime. It could be realized through the planning component of the MAPE-K autonomic computing loop. As a result, the way of adaptation mechanism represented in autonomic feedback loop should be explicitly [33] designed; because the control part is designed as a model-based entity. In selecting the adaptation strategies, it is important to consider the reason behind of adaptation. Examples of reasons include not satisfying goals that are specified in architectural

quality constraints (e.g., response time, throughput, energy consumption, reliability, and fault tolerance), behavior, undesirable events, state maintenance, anticipated change, and unanticipated change.

2.5. Machine Learning

Machine learning is an application of AI that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning emphases on the development of computer programs that can access data and use it learn for themselves. One of research direction to tackle unpredictable circumstances at runtime is machine learning for modification of adaptation space [31].

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide [34]. The primary aim is to allow the computers to learn automatically without human intervention or assistance and adjust actions accordingly. A more dynamic and cheaper solution is to use techniques such as those used in machine learning which can automatically update models so that the predictions of the model correspond to the situations that are actually observed at runtime. Machine learning algorithms are often categorized as supervised or unsupervised.

2.5.1. Supervised Machine Learning

It can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values, [35]. The system is able to provide targets for any new input after sufficient training. The SL algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

2.5.2. Unsupervised Machine Learning

USL is used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

2.5.3. Semi-Supervised Machine Learning

This learning falls somewhere in between supervised and unsupervised learning since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it or learn from it. Otherwise, acquiring unlabeled data generally doesn't require additional resources.

2.5.4. Reinforcement Learning Algorithms

It is a learning method that interacts with its environment by producing actions and discovers errors or rewards [36]. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal. In reinforcement learning, the learner is a decision-making agent that takes actions in an environment and receives a reward (or penalty) for its actions in trying to solve a problem [17]. After a set of trial-and-error runs, it should learn the best policy, which is the sequence of actions that maximize the total reward.

Because of learning strategy and ability in reinforcement learning [37], the authors understood self-adaptive systems can manage unpredictable circumstances by learning from a set of data or by selecting a best-so-far strategy from previous experiences.

2.6. Composite Components

The internal organization of a composite component is a data flow system that comprises of other components and data-stores. An internal component might connect its entrances/exits to the composite component's entrances/exits so that it could use the entrances/exits as its own to interchange data with the environment outside the composite component. Thus a software system could be modeled as a composite component. And a complex system could be constructed hierarchically from small parts as described in[33] . Additionally, some specific reconfiguration rules are required for component-based adaptation that describes and defines the attachment and

detachment of the components with the connectors in architectural configuration accordingly specified in [38]. So, the organization of those composite components and connectors in the self-adaptive system would make dynamic software architecture. According to the description of [39], dynamic software architectures describes state how architectural constitutes can be formed, interconnected and detached during the operational time to meet a specific goal.

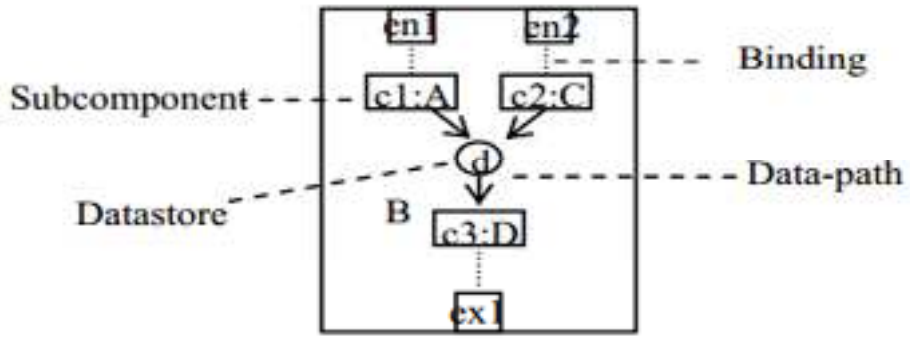


Figure 2.3 Composite component structures

For this reason, at a time of reconfiguration, composite components should have a capability of reconfiguring its internal components when the model of the architectural is constructed from it shown in figure 2.3 from [33].

2.7. Systematic Literature Review of Self-Adaptive System

In the reviewing part of the existing literatures that are related to this study, literatures are reviewed on the basis of some major question that is raised during the design and developments of architecture-based self-adaptive systems. The researchers discussed the question on the perspective of the literature and concluded the way how we introduced the question's answer in this work. The questions raised in this literature are supported in the work [27] and.[40]

2.7.1. Major Questions in Self-Adaptive Systems

I. What are the characteristics of existing methods in architecture-based self-adaptive systems?

The answers to this question tell us about the general overview of existing architecture-based self-adaptive systems with relevant characteristics of it. To answer this questions feedback loop organization, the component type that is subjected to reconfiguration, tools, and language for

development of the system and the reason for adaptation are the main attributes that need to be answered. Based on that:

A. ***Feedback loop organization*** identifies the number of feedback loops in each existing method. In adaptive systems, functions are composed in a feedback control loop that describes the activities in managing module in order to increase the reward and to predict the impact of the next actions before applying adaptation action over managed module [7]. The number of feedback loop would be single feedback loop [1], [8], [25], [27], [41], [42], two feedback loop organizations [26], and multiple feedback loop organization [28], [43].

   Using two or more feedback loop in self-adaptive systems will create more complexity during runtime; besides that, the level of entanglement between the functionality of reconfiguring resources and self-adaptation activities will degrade separation of concern concepts in architectural decomposition [18]. When the number of feedback loops organization evaluated on the angle of learning capability, based on that for each of learning strategies, single feedback loop design fashion is more manageable for lopping activities [5], [7]. Although some research papers [13], [30], [44] are categorized in an unknown group with respect to feedback loop organization, which are proposed learning strategies or algorithms to their reconfiguration action planning on basis search-based feedback loops [42]. This research work organizes a single feedback loop to tackle the complexity of reconfiguration at runtime.

B. ***Target Resource:*** This question investigates the concerned resources that are subjects for reconfiguration action; in another way what will the concerned resources or infrastructure that modifying its runtime behavior to fulfill some adaptation goal. Architectural component and connectors [1], [24], [26], [27], hardware resource such as movement in Robotics software [9], [30], [45], and services that provide computer and software resources to the customer and binding new service to the system [46]–[50]might be reconfigured resources to meet the desired quality goals. Additionally, the question also identifies the relationship of components or services that are being adapted/reconfigured itself to exhibit better quality. According to the categorization of reconfiguration resources, component-based reconfiguration in architecture-based has done with learning capability are [1], [9], [45], [51], that changes the topology of architecture dynamically which leads to change its architectural type and elements.

Those reconfiguration ways might be having direct or indirect impacts on system functionality and quality of the system. Besides that, it also determines the structural and behavioral characteristics of the system. Reconfigurations action based on based on architectural components will modify structural characteristics of the system if the adaption strategies are specified at design time without learning fashion. On the contrary, if it has learning capability it might modify its runtime behavior based on the previous action. The other issue on the reconfiguration of components at runtime is what strategies are needed if composite components are an element of the architecture which needs a big deal and high efforts to feasible the reconfiguration process on the existing infrastructure [18]. This work focuses on the reconfiguration of composite components because of in the modern ear systems are developing complex systems in a composite structure.

C. ***Tools and programming language*:** This question investigates tools and programming languages that are part of development methodology in architecture-based dynamic adaptation systems. Some research papers use used tools which are designed for general purpose application and others use developed tools that are specific and developed for specific research work. These identified tools and programming language in the existing literature are present in the table below:

Table 2.2: Existing tools in self-adaptive literature

Tools	***Description***
Music Studio	A suite of tools integrated together to help the application developer in creating adaptive applications based on the MUSIC middleware [24].
WEKA	a suite of machine learning software written in Java [24]
Autonomic computing toolkit	It is based on the Eclipse open-source framework, will work with the IBM Software Development Platform, and is designed to help developers add autonomic elements. [25], [42]
UPPAAL	UPPAAL is an integrated tool environment for modeling, validation, and verification of real-time systems modeled as networks of timed automata, extended with data types. [1]
OPERA	A layered queuing model used to evaluate the performance of web applications deployed on arbitrary infrastructures. [42]

PRISM	A probabilistic model checking tool, formal verification software for the modeling and analysis of systems that exhibit stochastic behavior. [7], [45]
MOFS	To develop transformation scripts for the transformation of the application variability model to Java source code [24]

Among tools PRISM is the most suitable for model checking that has stochastic behavior after reconfiguration in self-adaptive capability. Stitch is also used in many types of research for dynamic reconfiguration action because it has different modules that relate to tactics, strategies, and the impact of adaptation, rewards, modeling architecture, and environment activities.

Table 2.3: Identified programming language used by literature

Language	***Description***
Stitch	a language for representing self-adaptation plans within the perspective of an architecture-based self-adaptation framework [9]
Contextual Query Language	is a formal language for representing queries to information retrieval systems such as search engines [25]
Java	A high-level language which is based on object-oriented concept [8]
Python	It has built-in libraries for scientific python as numpy and matplotlib which strongly needed for ML[26]
DGCL	It combines programming concepts in a compact way, before the program is written in some practical programming language [52]
SQL	*SQL* is a standard language for storing, manipulating and retrieving data in databases [52]

It also provides a tactic and strategy module for an application called Znn.com which can reconfigure itself for performance-related quality.

D. ***Reason for adaptation*:** This question aims to identify the reason that triggers a reconfiguration process for tackling unpredictable circumstances at runtime is due to adaptation-goal or a side-effect of reconfiguration. Many types of research have published on QA based adaptation [4], [51], [53], [54], [9], [42], [46] are done based on service binding as the reason for reconfiguration processes, remains studies have mentioned functional changes are their reason for self-adaptation. In configuration action reason for adaptation has a direct relation to target resources that have described in question 1, B. if we combine these two attributes, 6 different attributes will exist that describes the reason of adaptation with the corresponding resource to meet a specified goal. These are; ***Quality-based and component*** [24], [26], [27]***Quality-based and hardware, Quality-based and service*** [47], [48], ***Service-based and component*** [42], [54].***Service-based and hardware*** [9],***Service-based and service*** [46].

II. How learning is introduced in the research studies?

The question investigates how the system learns from its environment to adapt itself accordingly as self-adaptive systems. The learning aspect of the system also needs some model representation in which abstracts the environments state as well as the architecture model state according to the problem domain. This review also states the mechanism to select alternative prioritization choices of quality or service in the decision-making process. The method of introducing learning to autonomic feedback loop would consider; which learning algorithm provides a solution for the proposed solution, way of model representation and design of quiescent state.

A. *Learning Algorithm:* Generally, the learning algorithms that are introduced into self-adaptive systems are divided into programmed fashion[2], [5], [6] which design and specified at design time by a programmer that might be anticipated at runtime time and machine learning that gives to a system learns from the prior experiences. The machine learning strategies grouped into two learning which are off-line learning [41], [51]- in which a system learns from labeled and example data are specified at design time, and online learning [31], [51]– in which the system can learn from its environment at runtime by exploiting the best-so-far experiences using machine learning strategies..

The fundamental concept in this question shows that machine learning with online planning strategy is clearly defined self-adaptive systems that over tackle unpredictable circumstances at runtime by using model-based reinforcement learning [1], [9]that consider current state, action, transition function and reward to consider whether the system needs to modifying its configuration topology at operational time.

B. ***Model Representation*****:** it uses to model self-adaptive systems at runtime for quantification analysis. Additionally, how problems are abstracted into a runtime model that represent environmental and architectural element also the basic consideration in model representation. For this reason, the environmental model represents all states and resources that interact with systems that need to being reconfigured which include connector and components [43]. To make this abstraction of the problem in model representation of systems, fitness calculation and operator with learning process application and reward calculation for next learning and decision making are performs in design space of self-adaptive system [9].

Among multiple kinds of model representation ways existing in previous works use QA data variable [42], [43]which represent the characteristics of multiple form of single quality attribute, Utility function [2], [7]to reason about decision making for reconfiguration action selection from alternative planning strategies in weighted function quantity, Markov chain [7], [9] for model representation that can make stochastic behavior analysis at runtime after reconfiguration, are used repeatedly. These models representation has a direct relation with the learning feature that the system brings to make a decision on the selection of architectural configuration.

III. How are the decision making and tradeoff analysis of multiple QAs performed at runtime during reconfiguration?

The decision making and trade-off analysis process also has effects on choices of assurance techniques, how data is monitored and gathered for analysis in the feedback loop, how feedback control strategies organized and decision on selection of change management model styles for proposed in the self-adaptive system. Four different issues are raised for decision-making and trade-off analysis considerations:

A. ***Assurance way*****:** describes the method that provides any evidence for validation the result of decision-making mechanism regarding satisfaction of adaptation planning after adaptation. Assurance types in which different studies have conducted are: Simulation [26], [31],is using in

self-adaptive systems that uses hardware resource as a major subject of reconfiguration, runtime verification [4], [13], [24] and model checking [1], [5]also kinds of assurance that calculate quantification of the reconfigured architecture and making stochastic behavior analysis with the impact and reward calculation respectively. The remains researches are using testing procedure [43], [48], which accomplished through human-driven evaluation of self-adaptive systems. In model checking assurance, machine learning has crucial places since it makes stochastic behavior analysis using Markov decision process [9]. However, if failure recovery techniques available at assuring time, the system should have to shut down itself and restart to maintain the failure based on the work [21]

B. ***Handle exploitation-exploration trade-off analysis*****:** question identifies the way how a self-adaptive system gathers information and how planning selection is going to be selected from knowledge. Exploration [44], the system makes trial decisions to enrich its knowledge and exploitation [4], [24], [48], the system utilizes its experience to maximize the expected reward. Some papers used exploration-exploitation strategy by managing the tradeoff between them using e-greedy method [1].

C. ***Adaptation control unit*****:** it poses how to realize adaptation of a software system using control loops for decision and interaction. If a single component realizes adaptation in a whole software system, the adaptation control unit will structure in a centralized manner [6], [8]. In turn; if multiple components realize adaptation by themselves i.e. the overall reconfiguration behavior arises from a localized single component it is called decentralized control unit [18], [41].

D. ***Reconfiguration pattern*****:** This answer defines the different solution in pattern form for interacting decision control loop in the complex self-adaptive system [55]. This pattern considers different interaction among different phases in the feedback loop on the basis of the MAPE-K loop. Based on that [24] study left this step to software developer and [46] also left the reconfiguration process to the software developer. The other else didn't mention specific reconfiguration pattern for describing the relationship in the phase of a feedback loop for their proposed work.

2.7.2. Reviewed Discussions on The Literature Review's Result

QA based adaptation for architectural elements reconfiguration is suitable: software architecture has direct impacts in software quality attributes. The structural and behavioral characteristics of

architectural elements and their relationship will determine extensively the level of quality that the system would achieve. QA based adaptation is the main reason behind for dynamic reconfiguration action to hinder uncertainty at runtime. In every reconfiguration of components and connectors, the system would deliver some impacts on its non-functional requirements of the system.

***Achieving dynamic reconfiguration in the composite component is complex*:** if dynamic reconfiguration is performed in simple type components the reconfiguration process might be easy because it would maintain the concept of separation of concern easily. However, in a composite component in addition to reconfiguring component, reconfiguration capability also needed for the internal components that would create entanglements of functionality and adaptation action. The answers of research question 1B also confirm that no research works have done that performs dynamic reconfiguration on the basis of composite components using machine learning capability.

***Aspect-oriented based self-adaptation*:** in the programmed fashion based adaptation most of the studies use aspect-oriented software engineering mechanism which separates the crosscutting of the software architectural component into aspects [2], [13]. After the developer set high-level abstraction, modeling business goal and specifying reconfiguration related logic that is the main part of the development process that can directly translate the models of the architecture into executable code by using code generation tool. This paradigm has focused on changing the part of the code without stopping the execution of the system called dynamic weaving as described in [56] study.

Lack of dynamic architectural description language: to simplify the development and discovering solution in dynamic reconfiguration process, the researcher/developer should tackle the design space challenges as a roadmap of software engineering for self-adaptive systems. To formalize the decision of control unit, solving the tradeoff among multiple decision choices of quantification and qualification [4] should be clearly described during the design space-time using architectural description language.

***Reinforcement learning for architecture-based self-adaptation*:** reinforcement learning is a powerful machine learning types which are based on the feedback values of the agent. Other kinds of machine learning which are supervised and unsupervised learning need a set of examples or historical data to predict the output of the next action. But with the absence of full data, the system can't predict the consequence for some unpredictable circumstances which may exist at runtime. In the case of reinforcement learning, the system selects its adaptation strategy based on feedback

analysis. As answer to question 2A, among the primary studies that use machine learning, most of the studies use reinforcement learning strategy.

***Lack of reconfiguration pattern and quiescent state description*:** in most of the primary studies the reconfiguration pattern which describes how a set of components that make the software/system architecture collaborates to change system architecture to new one based on a set of some reconfiguration command not described at all. So, reconfiguration action or sequences of action that performed under specified reconfiguration pattern will simplify the development process in a self-adaptive system [27]. Similarly, the quiescent state is a place in which dynamic reconfiguration has taken place. The main issue raised in this idea is how this place designed and what looks like its feature before and after adaptation. It is possible to say, in all primary studies there is no description of the quiescent state.

2.8. Related Works

To present a clear understanding of general methodology and characteristics of the architecture-based self-adaptation process, the researchers discussed the most related works to this study. Accordingly that:

In MUSIC [24], The framework provides very comprehensive and effective development support while also removing a substantial amount of work from the shoulders of the developers. The tight alignment of the methodology with the middleware works very well. The flexible plugin-architecture of the context and the adaptation middleware shows clear advantages when individual applications require their own specific solutions (e.g. specialized Context sensors and context data analysis, communication protocols, or adaptation reasoning). The integration into Eclipse has proved successful and effective. The MUSIC demonstrators are helpful in getting acquainted with the framework and help serve as blueprints for new applications.

In FUSION [57], present an empirical evaluation of the framework using a real-world self-adaptive software system. It provides a comprehensive description of a black-box approach for engineering self-adaptive systems decisions are made using abstraction. It proposed solutions using online planning-based model-free reinforcement learning. Self-adaptive software systems build on the notions of feature-orientation from product line literature. It combines feature models with online learning. The proposed framework copes up dynamically changes that are unforeseen at design time

using OL (discrete value). It can make runtime analysis and learn feasible. A feature in this approach is realized as an extension of the software architecture at well-defined variation points. It fits in three-layer reference architecture. FUSION is fitted in the goal-management layer. The study [26] uses XTEAM in change management layer as well as Prism-MW for control component layer.

In ACTIVE FORMS [1], DevOps with self-adaptation concepts executing the formally verified model of the feedback loop by a virtual machine. The model provided guarantee for the adaptation goal by using model checking at runtime to select a configuration for adaptation. Updating feedback loop models to deal with changing adaptation goals or adding new goals. The approach is divided into design/evolution time, deployment time and runtime which consist the corresponding entities are stub models, external components and managed system respectively. The main function of each entity are model verify feedback loop by templates model checker, deploy feedback loop and active forms runtime environment and collect additional knowledge, verify the quality of services & Adapt managed system by the statistical model checker. It used networks of timed automata (TA) for the specification of feedback loops and timed computational tree logic (TCTL) to specify properties. It shows the experiment on IOT building security infrastructure on energy conception and packet loss adaptation goals.

In GRAF [8], adaptation at the level of fields and methods, model-centric runtime adaptively approaches. Software system to be controlled at runtime considers as base-layer in reflective architecture. Adaptable software connected to meta-layer which is an actual runtime model acting as mediator. Adaptation manager can control the managed system through runtime model rather than to control directly. Adaptable software changed by either parameter adaptation in a means of changing variables values, or via compositional adaptation through adjusting the software's structure. Integrate the runtime model as deeply into the software as possible. The idea is to build generic software based on runtime models that describe the variable parts of the system. The main way of achieving adaptively in this research is by redirecting the adaptable software's control flow to a model interpreter component at points where the need for adaptively is expected. Reflecting changes to and from the runtime model by using code injection mechanisms of AOP by identifying the adaptability factor a new program can be introducing and/or transforming into adaptable software.

In Han Nguyen Ho [18], 2015, proposed a model-based reinforcement learning solution for planning part in self-adaptive systems. For decision-making purposes, Bayesian-based RL is by formulating belief states that is the cross-products of all states and the model parameter for representation of uncertainties at runtime. Computational complexity is the major drawbacks of the model-based RL in the study. The drawback is raised from performing learning state due to the action selection process and predicting the next state and action. Because of that, minimizing the selection space by using utility theory would be minimizing computation complexity problems.

Together the above all studies provided important insights into the designing and development of architecture-based dynamic self-adaptive systems. On the basis of the above all, the researchers acknowledged that the combination strategy of MBRL and UB enables self-adaptive system learning from its environment in a better manner. Similarly, [58] found that if the model of the self-adaptive system accumulated more previous knowledge before decision-making, MBRL algorithm would be select best strategy that can produce with the highest reward values. In contrast to [9], [17] argues that due to the performance degradation behavior, MFRL algorithms have more efficiency with the absence of online learning strategies. Conversely, [59] reported no significant differences between MFRL and MBRL if the strategy of action selection is balanced as exploration-exploitation tradeoff mechanisms. Almost every paper that has been written on ABSA includes the MAPE-K reference model relating to Rainbow framework [2], [23]that helps us a foundation for this study. The evidence presented in the literature [60] on SAFU values helps us to calculate the effectiveness of our proposed framework against the existing ones. The above all related works are also used MAPE-K feedback loop for controlling part of adaptation based on the principle written in the study [61]. We also suggested the organization of the MAPE-K feedback loop in dynamic software reconfiguration should structure as similar ways with the above-related work.

As the researchers understand at the all, mixing of formal method or protocols described in research roadmap, reinforcement learning strategies with controlling the priority of multiple reconfigurations and human interaction for testing should be considered in the proposed framework. This understanding also justified in the [62], which explains each strategy in the design of the autonomic framework.

2.9. Summary

The researchers recognized that the most architecture-based self-adaptation approach uses a single feedback loop in a different structure that realizes MAPE-K functionality. Since the self-adaptation process is based on architecture-based, it uses architectural elements includes component and connectors to realize dynamic reconfiguration to hinder the complexity after adaptation. PRISM and Autonomic computing tool are the most tools used in the model-checking of a self-adaptive system. However, stochastic behavior analysis is the simplest way of model-verification ways in autonomous self-adaptive systems.

Furthermore, as the researchers' reviewed results shows that mapping the architectural configuration to environment state needs an entity that anticipates changes at runtime. Therefore, there should be a strategy that specified either in programmed fashion or machine learning strategy. Both have an equal contribution to this reviewing result. Markov chains and utility function has the highest priority for model representation of the system. The Markov chains work in the principle of discrete time Markov decision in reinforcement learning. In Utility function part it uses quantification of alternative strategies to select the best reconfiguration decision on each of quality goals.

Finally, in the structure of feedback loops, the authors understood the centralized control unit is the more recommended approach in primary studies which is the most widely used mechanism. That's why the control unit also uses exploitation for monitoring the environments of the systems and selection of primary strategies. In parallel, testing is used mostly for checking the system after reconfiguration on the basis of model-checking processes.

CHAPTER THREE

3. Research Methodology

The architecture-based self-adaptation methodology is familiar with providing a solution for the autonomic computing system that has dynamicity characteristics to reconfigure itself at runtime. The methodology includes the following methods: feedback loop designing and developing [3], organizing architecture model and environment model in system that can communicate through well-known interface, deciding reconfiguration pattern with control strategy [11] and finally building learning components which composed previous experiences that enable a system adapts itself accordingly based on machine learning strategies. Before that, design space decision [63] should be considered in the methodology part. Some other activities such as: updating the learning component and executing feedback loop components (MAPE-K) should be decided at runtime.

The study is generally followed by architecture-based self-adaptation approaches on the basis of the following methods.

3.1. Systematic Literature Review

The researchers conducted a systematic literature review to identify the characteristics of self-adaptive systems and to refine the state-of-arts as well as the knowledge scope in the ABSA trends. The systematic literature review was prepared according to the procedure used by [27] because of the decision dimension of the self-adaptive system and the requirement to design it reviewed accordingly in the paper. The authors used the search process of [27], which designed in four-phased searching steps to collect the relevant studies. These are Manual search phase, Automatic search phase, filtering phase, and a data collection phase.

The search string consists of four parts, Dynamic **AND** Reconfiguration, **AND** Architecture **AND** Machine Learning

The main search string is done by connecting the following keywords through logical OR as follows

> (Self **OR** dynamic) **AND** (reconfigure **OR** reconfiguring **OR** reconfiguration **OR** adapt **OR** adaptation **OR** adaptive) **AND** (architecture **OR** architectural) **AND** (learning **OR** machine learning **OR** reinforcement learning).

To do the automatic search, the researchers used the most relevant electronic sources to software engineering research areas as listed below:

1. IEEE Explorer
2. ACM digital library
3. SpringerLink
4. ScienceDirect

These libraries are the most appropriate to the field of self-adaptation study and they provide most of the selected venues. They also provide suitable and easy search engine mechanisms which are simple for the spontaneous search of resources in their database based on the combination of search term/string.

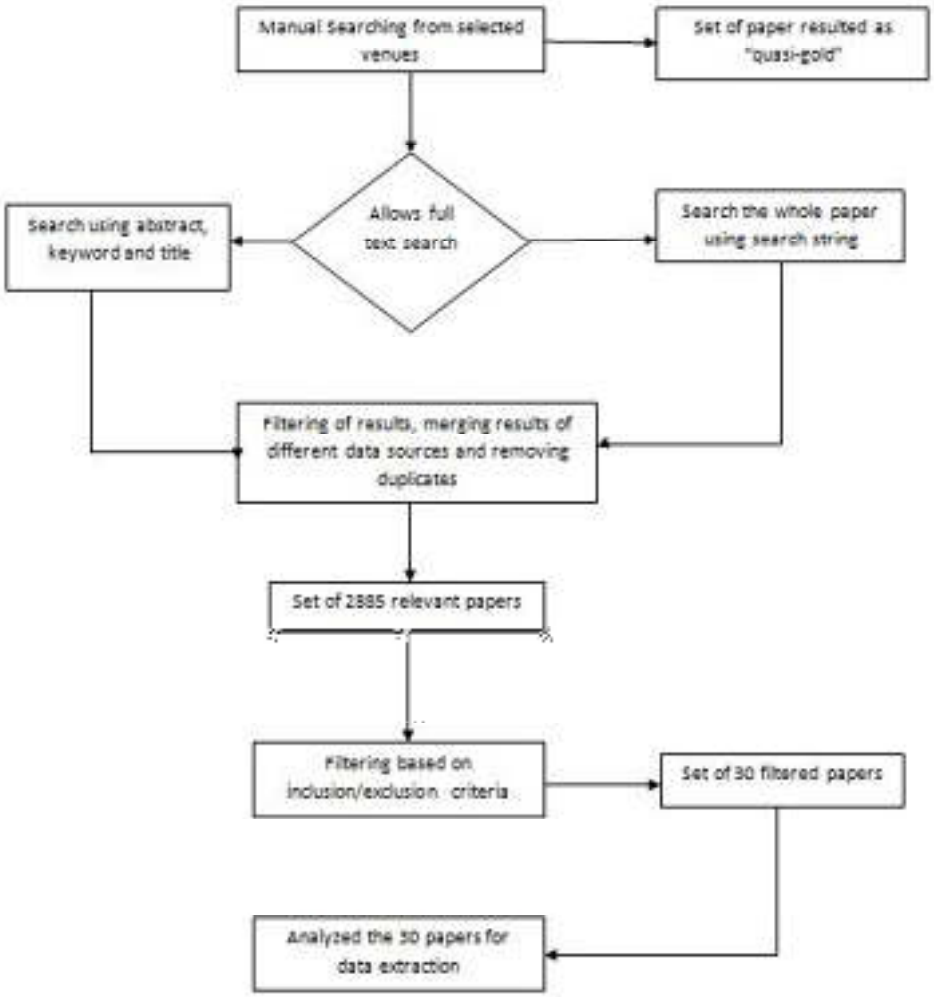
3.1.1. Refining the Search Result

Inclusion Criteria

1. The study should be in the area of self-reconfiguration in a dynamic manner. The system also should have learned from is the environment and change its stricture using its external model.
2. The method proposed by every study should be architecture-based to over tackle the new unpredictable circumstance which exists at runtime.
3. The self-adaptation processes have accomplishing to satisfy different quality attributes in other means the reconfiguration steps should handle multiple quality attributes at runtime.

Exclusion criteria

1. A study is not written in the English language.
2. A publication year is not between 2009 to 2017



3.1 Systematic literature review path approach

3.2. Proposed Frameworks

The proposed frameworks are the combination of the rainbow framework with learning and utility performer additional components based on the MAPE-K autonomic computing loop.

It consists of developing architectural and environmental models for abstraction of the system into a model that can learn from environments, which enables a software system modifying itself at runtime.

The followings are some of the specific methods that help to feasible the whole proposed methodology for better understandability.

Method to feedback loop arrangement: the number of MAPE-K loop in the research work. This work uses a single feedback loop. However, two and multiple numbers of MAPE-K loops can be designed in one system with large complex activities [26]. The arrangement also described within model-based fashion.

A method handles the reconfiguration process: this describes how architecture constitutes such a component and connector makes communication and/or collaboration. The process should have followed specific reconfiguration that is described in [11], which is a client-server reconfiguration pattern.

Method to handle multiple configuration trade-off: to decide the best alternative configuration for the current state with high reward, the system's model should exploit previous experiences from knowledge component or explores solution in trial and error. For this reason, utility function will uses for quantification of the current architecture state to the current environments if two or more rewards are selected at the same time t.

Method for controlling strategy in a feedback loop: describes how managing component's MAPE-K loop controls the environment of the system and makes decision accordance to it. This control strategy organized in a centralized control pattern style, in which the whole component of the feedback loop will mapping in a single model.

3.3. Prototype Development

A case study approach was used to illustrate the proposed framework that helps in evaluate the solution by comparing with the existing frameworks. It maintains the approaches described in [7][9][60] study. For that matter, IntellijIdea tool serve for development because of its functionality by integrating of java and python language. The aforementioned tools and programming language stated in section 2.7.1 has shortcoming in integration of them.

3.4. Stochastic Behavior Analysis

Since self-adaptive systems reconfigure themselves according to the environment, they would exhibit stochastic behaviors. To analysis this behavior, Discrete-Time Markov chain (DTMC) based on Markov decision process that considers state, new state, action, transition probability and rewards for the reliability of the system. This analysis has aimed to check the model consistency after reconfiguration whether the newly reconfiguration results has confronted with the functional

and non-functional requirement of the system specified at design time. So the way if determined to analyze the impact model is seen in section 4.7. It was considered that quantitative measures would usefully prioritize multiple configurations and helps to check the consistency of architectural constraints and functional requirements.

3.5. Evaluation

The whole methodology and proposed frameworks will be evaluated based on the output result. It will be presented as their performance and reliability efficacy against a number of episodes and the conclusion will be made on the dynamic reconfiguration of composite components using reinforcement learning.

Prototype evaluation metrics way:

- Different reinforcement learning algorithms with their action selection strategies.
- Utility output of each adaptation strategy with the quality preference conservation.
- Performance and reliability measurement on collected metrics on each adaptation ways.
- Actual reconfiguration state and action result vs. with predicated actions and state.

Additionally, to further elaborate the result of this study, the researchers used self-adaptive fitness unit measurements. The SAFU approach has a number of attractive features that would help evaluates self-adaptive systems easily as explained in the work [46].

CHAPTER FOUR

4. Proposed Solution of Self-Adaptive System

4.1. Proposed Framework for Solutions

To provide solutions for the research problems listed in 1.3, DSARCC (Dynamic Software Architecture Reconfiguration in Composite Components) framework has developed to mitigate the identified problems. DSARCC is a comprehensive self-adaptive software development framework that satisfies the working principles of both MAPE-K feedback loop and reinforcement learning by overlapping one to the other with the integrating of the utility theory concepts. The framework adds learning capability for its self-adaptation process by changing the architectural topology dynamically accordingly to the environment. The learning part of the components uses a model-based reinforcement learning approach. A development framework extensively supports the self-adapted application with a module that can integrate model checking tools. Besides that, the framework also will provide

- ✓ Separation of control activities (managing component) from the function of the system that is being adapted itself contextually according to the environment changes (managed component).
- ✓ Stochastic behavior analysis component that enables the system to make runtime verification for the impact of the model using DTMC as a probability model checking tools.
- ✓ Uncertainty analysis in both utility and reinforcement learning that defined both at design time and runtime.

As fig 4.1, shows that the framework has two main parts. Managing system and managed system. The managing system is organized using MAPE-K feedback loop that controls the autonomous self-adaptive process of the managed system. Each parts of the DSARCC would be discussed briefly in the next section. The figure shows the whole proposed framework with the middleware infrastructures that are adopted from Rainbow. The framework provides some solution which can solve the drawbacks of the Rainbow framework and can be also applicable in architecture that constitutes composite components.

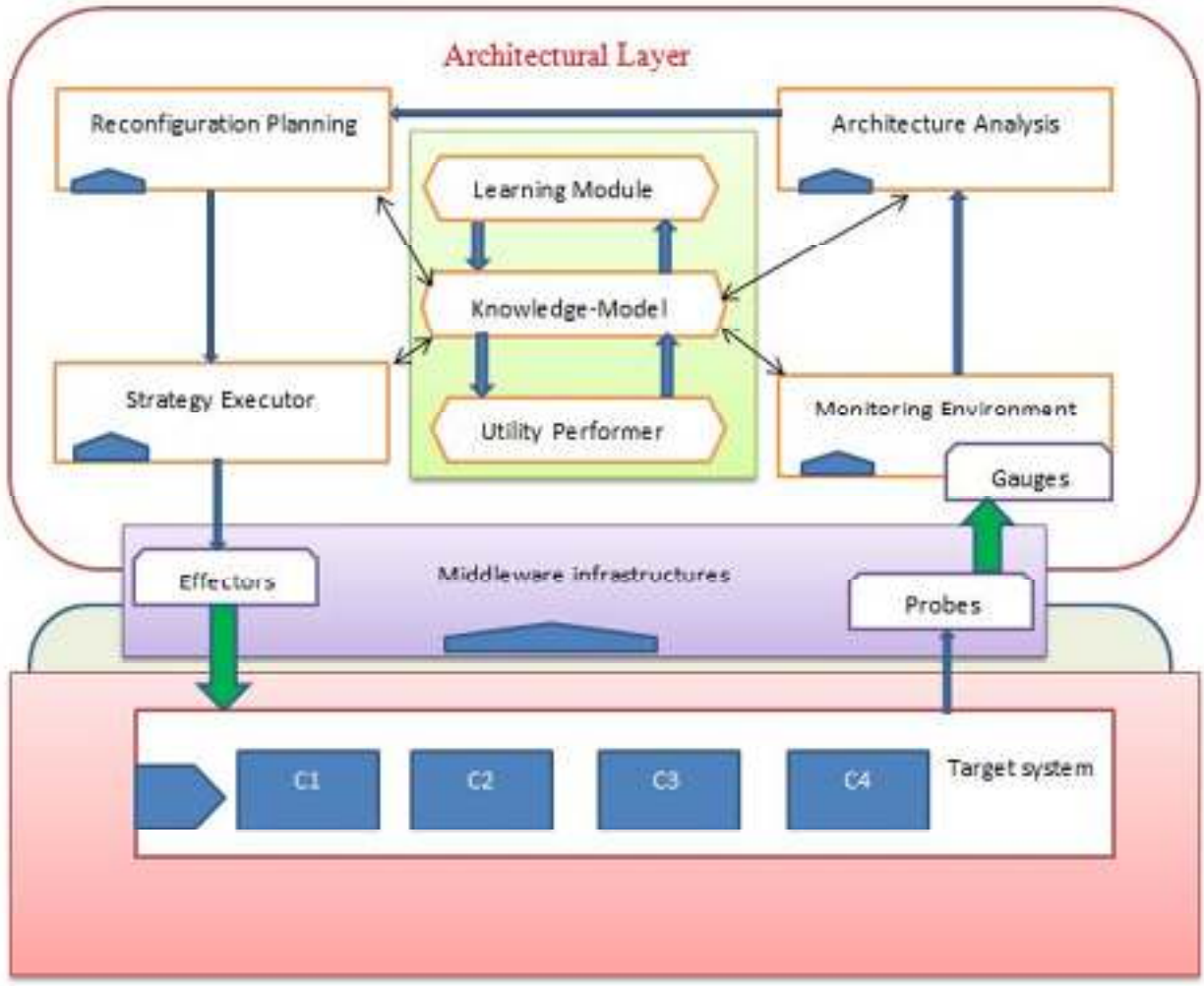


Figure 4.1 Proposed Framework for a solution

4.1.1. Managed Components of DSRACC Framework

This subsystem is composed of the functional component of software systems that needs to adapt/reconfigure itself dynamically. In the prototype, the architecture of the system is organized in composite components. Functions will be embedded in other function as a form of components. This part of the framework is subjected to reconfigure its architectural topology at runtime. It will be demonstrated in Chapter five case study parts.

4.1.2. Managing Components of DSRACC Framework:

Mainly this subsystem is composed from feedback loop that controls the managed system state and performs reconfiguration based on the learning strategy. It emphasizes the arrangement of MAPE-K autonomic computing loop with a learning model. As the figure 4.1 shows that it consists the general MAPE-K components with leaning components and utility performer.

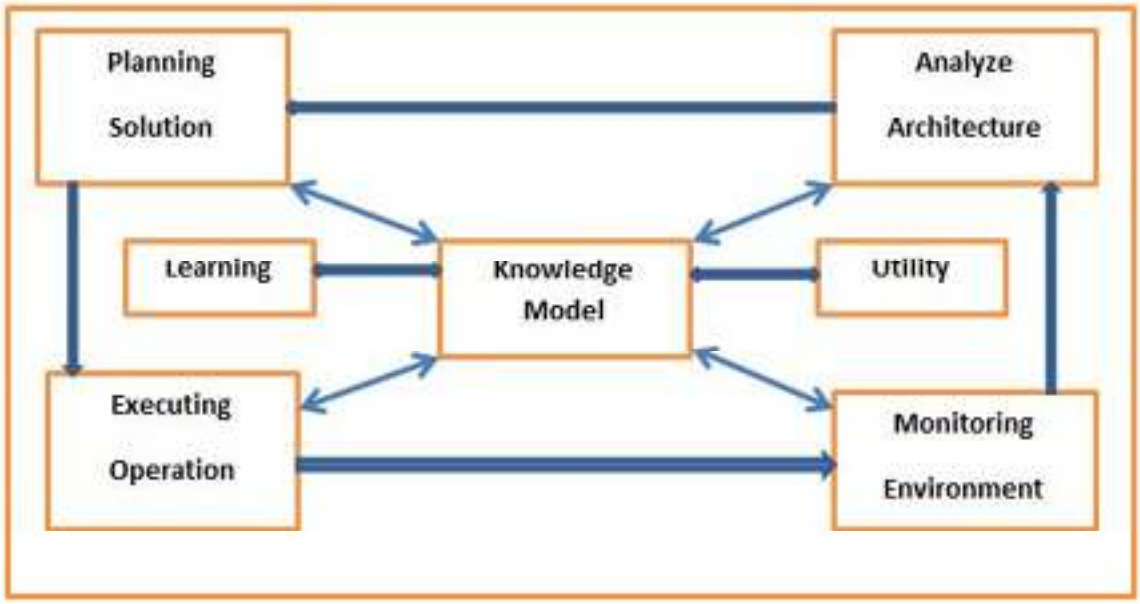


Figure 4.2Managing system in the DSRACC framework

4.1.3. Feedback Loop (Control Loop)

Control decision dimension has realized by the interaction of feedback loops in the framework. It is responsible for measuring changes that are introduced into the system at the operational time. Concurrently, by some techniques, there should be a determination of the number of changes to computing how much change misbehaving from design decision issues. The main decision concepts about dynamic self-reconfiguration system's control are: "what are the involved control loops?" and "how do those control loops interact?" the composition of feedback loops with the structure and complex the system determines the choices of involved control loops. DSARCC is structured using single feedback loop organization for avoidance of entanglement between the components multiple control loop functions that leads the system more complex. The control strategy and algorithms with the working principles are generalized in every description of the MAPE-K control loop with a learning component.

In the control decision dimension, what part of the system being adapted that can undergo to another configuration state are also key issues. Thus the adaptation operation would be changing a parameter, addition, and deletion components attach and detach connectors. Controlling the introducing of architecture-based adaptation process needs some entity that is organized as a centralized or decentralized pattern. As discussed in the literature review, the controlling entity is organized in a centralized pattern form in the proposed solution. Since the learning process is model-based reinforcement learning, the model is prepared as external control entity inside the managing system. In this part, the software architecture will consider as model-based RL agent as an architectural model.

The feedback control loops work as follows in general steps

```
Pseudo code:
   Step 1: collect runtime metrics
            int getCompProp(int Q1, Q2, …, Qn)
                   getQ1();getQ2(); …;getQn();
                   return Q1,Q2,…,Q3;
            abstract int getCompProp(int Q1, Q2, …, Qn
                   getQ1();getQ2(); …;getQn();
                   return Q1,Q2,…,Q3;
                   aggregate(Q1,Q2,Q3,…,Qn);
                   analyze();
   Step 2: analyze the architecture
            void analyze(int Q1, Q2, …, Qn)
                   gauges(Q1,Q2,Q3,…,Qn)
                   checkCondition();
                   if(archConstraint!== aggregate(Q1,Q2,Q3,…,Qn)
                      planning();
                   else
                      go to step 1
     step 3: Planning a new configuration
              int planningConfig(curState, strategy)
                     int performUtility(int Q1,int Q2,…,int Qn)
```

```
                    return Utility(curState, strategy)
                int performLearning(int state, int action)
            return Learning(state,action, reward)// exploration
                    if(reward > curReward)
                    updateModel();
                    else
                    selectbestsofar;
                int perfromDTMC(Model model)
                    return model;
            executeModel(model);

step 4: execute the new model
            int executeModel(Effector model)
                return model;
            getCompProp(Q1,Q2,…,Qn);
```

4.1.3.1. Monitoring Environment

In order to collect runtime metrics about the managed system and passes the aggregated result of gathering information for the next phases of the cycle, the information would be collected using a component called Probes. Since the self-adaptation process focuses on the autonomous component-based software system, all information should be gathered by probing through the object of each component in managed system. States of class's variable, objects components, new requirements for functions, the structural topology of the architectures all are to be continuously detected by the monitor components. Because of that, this part enables the managing system to understand managed systems state. It also directly mapped to the Monitor components of MAPE-K autonomic loop. As a result, the information captured from managed systems through probes is translated to monitoring components using gauges. It maintains a common protocol between the components of managing systems and managed systems.

To monitor the environment of the system, representation of the system's environment is very crucial according to the design space dimension in self-adaptive systems. This representation should be described in goal-scenario fashion at design time for better understanding.

The goal-scenario in the system is restructured as indicated in Table 4.1.

Table 4.1Goal and Scenario in Self-adaptive system

Goal	***Scenario***	***Condition***	***Reaction***
Maximize the execution time of the system	When the number of multiple process increases, add an extra server to delivering output and/or convert the delivery to Image	Number of process increases	The extra server will be added, and/or the delivery output will be in Image
Minimize the cost	When the number of process exit, down some servers and/or convert the delivery to video format	Number of process down	Server down, and/or Convert to video format
Goal -3	Scenario-3	Condition-3	Reaction-3
Goal-n, m	Scenario-n, m	Condition-n, m	Reaction-n, m

The above table shows multiple reactions to a specific condition. These multiple reactions might be applied based on the utility priority calculation by selecting an individual strategy or both on the basis of trade-off results. The condition column in goal-scenario [17], to enable performing of transitions between states, considers as uncertainty or unpredictable circumstance during at runtime, but this proposed solution considers by questioning reliant interactions between functions or components, the study essential to take in to account other factors of internal structure/implementation [9] that might have an effect on the adaptation action.

Additionally, the probes and gauge part of monitoring components used for aggregating and abstracting runtime information on about the model of the environments [1].

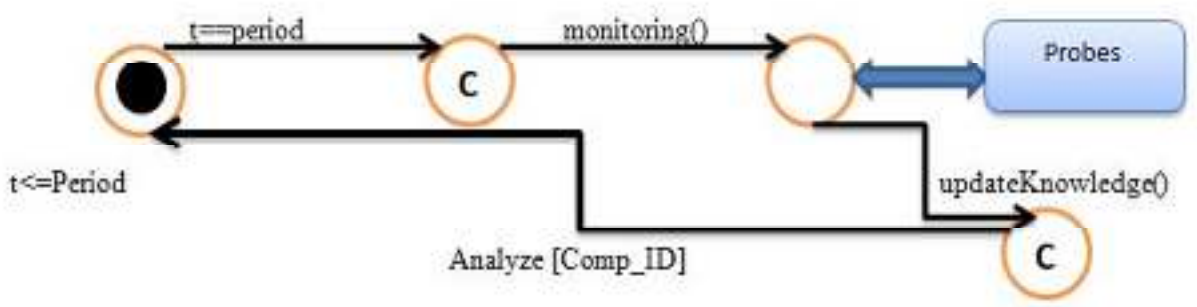


Figure 4.3Monitoring task ways

A time generated Monitor periodically (t==Period) stimulates a probe via monitor(). If necessary, the newly gathered data is preprocessed. Then the Knowledge model is updated (update Knowledge()). Finally, the Analyze behavior is informed about the update Analyze[Comp_ID]. This latter triggering is optional, as the Analyze behavior may use time triggering.

4.1.3.2. Architectural Analysis

This component of the framework is like Analysis components of the corresponding MAPE-K loop. It analyzes the information from monitoring environment components based on the knowledge stored from the previous experiences gained by learning capability. It decides whether adaptation is required or not to satisfy the current environment. Beside that the component also performs quantification of the current architecture configuration to the current environment to decide the rewards of reconfiguration and predict the output. It determines the rules of architectural constraints when the model property has been changed because of environment state changed. Thus it helps to evaluate the conformance of new architectural reconfiguration to the system's architectural constraint based on specified situations.

Hence, Analyze behavior matches the required resources with the resources in use; it is taking into account the current context and working data. Analysis can result in three primary conclusions. The current situation is satisfied when the managed system possesses the resources it requires to realize its goals. The situation is under satisfied when the system lacks resources, and over satisfied when the system has redundant resources.

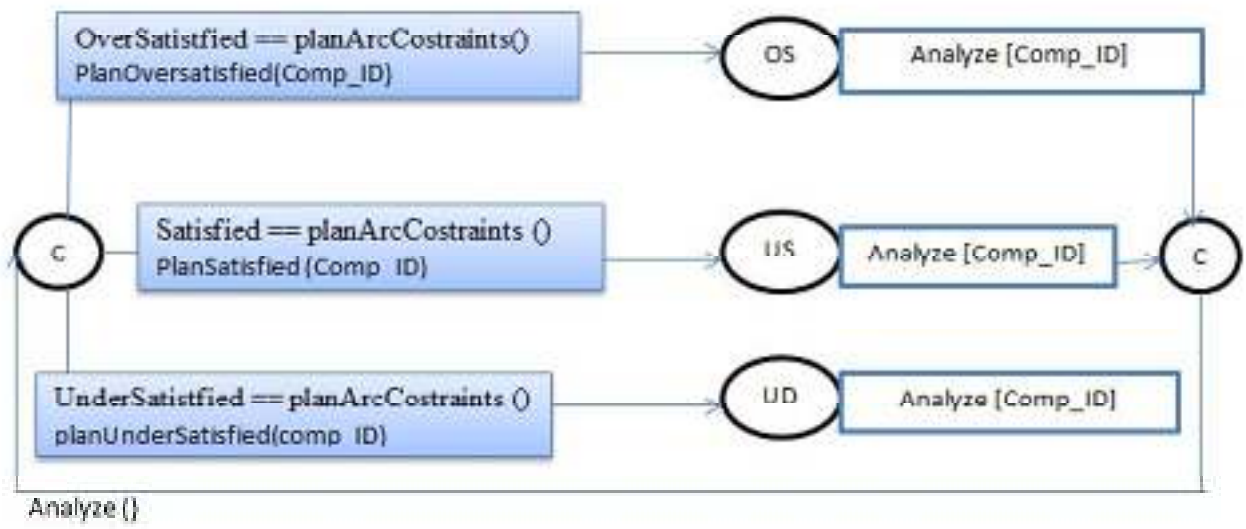


Figure 4.4Analysis path

4.1.3.3. Reconfiguration Planning

It chooses a new architectural configuration based on the information from the architectural analysis component. This part also maps to planning activity. When the architectural analysis triggered adaptation, reconfiguration planning component will selects new architectural configuration based on the best strategy that is calculated by the decision-making strategy. During this stage, the system must consider some generic question in the self-adaptation process. Such as: **what to adapt, *when to adapt, and how to adapt?***

The Plan behavior is responsible for planning adapting actions when needed to bring the managed system in the desired state (i.e. Satisfied). There are two cases that require planning: either there is a need to add servers (i.e., the system state is in an unsatisfied situation), or servers may be deactivated (when the system is in an over-satisfied state). The authors present two reusable patterns for Plan behavior. With time triggering, the planner periodically controls what it needs to do (x==planToDo() condition), and makes a suitable plan when needed (ReleaseResource or AddResource).

Since the combination of Utility outcome and RL is proposed as decision-making strategy, the major questions is raised would be, which algorithm should be executed first and serves as input for the others. The work [7], address planning new configuration by combining utility theory with a

stochastic model of the strategy. It provided a method to quantify strategies relative to the objectives, under uncertainty. So DSARCC uses utility theory for prioritization of the strategy because of its static behavior analysis.

Algorithm 1: Utility theory

Input: all utility values

Output: utilities of all strategy

3.7.1. Begin:

3.7.2. Loop given all utility values of each quality to each strategy, utility = 0

// switch to the highest utility with qualities to state and action pair

3.7.3. Do i to n times

3.7.4. Utility(Quality) ← Utility(function + Impact)

3.7.5. Utility(Strategy) utility + Utility(Quality) * preference

3.7.6. Until all utility of strategies has calculated

3.7.7. Until all utilities, if qualities under each strategy have calculated

3.7.8. (state, action) ← utility

3.7.9. end

The algorithm will return the strategy as action and qualities with their corresponding utility values. This result will form a single state-action set and passed as a parameter to reinforcement learning. Consequently, exploration time of the learning process will decrease due to the admissible state space as an operator in [17]. The above algorithm uses the utility theory concept by generalizing utility function, utility preference and impact vectors which discussed briefly as follows.

4.2. Utility theory

As working in Rainbow, for decision-making algorithm, the research work proposes utility theory for quantitative verification method of runtime. Once environmental and architectural information has collected and the problem is detected, there would have to choose the appropriate new architectural configuration. To detect misbehaving adaptation against the system functional requirements and the overall business constraints, there have to capability to consider multiple alternative adaptations that have different possible effects on the system. E.g. an adaptation that fixes reliability issue might affect performance concern or vice versa [26].

To choose the most appropriate adaptation strategy for a given changed environment, there must be defined values that express system objectives to tackle some uncertainty which happens due to unpredictable circumstances such as incorrect detected information, lack of sufficient learning time, or unanticipated error at the runtime. Therefore, to address this problem this study will combine the utility theory with stochastic behavior analysis like most of the works around self-adaptive systems. The work [7] explains the utility theory in three different values:

1. ***Utility function***: it measures the collected system properties for providing values which show how reconfiguration strategies are behaving when searching to achieve the new quality goal. This function also defined in deigned time phase.
2. ***Utility preference***: it defines the relative importance of the quality dimensions and serves as an example to prioritize quality attribute for the intended reconfiguration pattern. It can also solve the limitation of systems resources or trade-off between certain qualities attributes.
3. ***Impact Vectors*:** the impact of strategy on each of the quality dimensions represents as a vector of cost benefit-values between the strategy and each quality dimension

The above three values are called utility outcome. They serve as input to calculate the utility rate for ranking adaptation strategy through the utility outcome of Equation 4.1 as adopted from [7] research study.

$$UQA = Utility\ (QA + impact\ (value))$$

Equation 4.1*Utility of the quality goal calculation formula*

Where UQA is utility outcome of specific quality attributes, **QA** represents quality attributes, **Impact(value)** represents the impact values of specific QA on systems which are specified at design time.

U_{Q1} = Utility(UQ1 + ImpactVector(Q1))

U_{Q2} = Utility(UQ2 + ImpactVector(Q2))

U_{Q3} = Utility(UQ3 + ImpactVector(Q3)) … U_{Qn} = Utility(UQn + ImpactVector(Qn)

$$Utility = U_{Q1} * preference(Q1) + U_{Q2} * preference(Q2) + U_{Q3} * preference(Q3) + \ldots + U_{Qn} * preference(Qn)$$

Equation 4.2 Total utility calculation of action

4.3. Learning Strategy

This model abstracts machine learning algorithms, strategies, tactics which gives learning capability to the system. Because of this, the system can learn about the environment and predict the impact of reconfiguration simultaneously. The strategy for choosing new architecture configuration is a stochastic process in which the system exploits previous knowledge [9] or explores a new action [44].

4.3.1. Reinforcement Learning

Formally, reinforcement learning (RL; Sutton and Barto1998) describes a type of solution to Markov decision process (MDP) problems which are defined by a tuple S, A, T, R, and π:

- S: a set of states' $s \epsilon S$
- A: a set of actions $a \epsilon A$
- T(S'|s, a): the transition function maps each state-action pairs to distribution over successor states s', with $s, s' \epsilon S$; $a \epsilon A$, and $\sum_{s'} T(s'|s, a)=1$
- R(s, a, s') $\longrightarrow$ r: the reinforcement function mapping state-action-successor state triples to scalar runtime r called reward.

The corresponding attributes of MDP tuple in the framework implementation would be assigned as the following for the proposed solution:

- S: a set of states in which exists in system execution and are a {execution time of the specific function, mode of delivery, amount of resource consumed by the system at run time, amount of resources available at runtime}
- A: a set of actions are {reconfigure component: attachCom, detachCom, addServer own server, setImage, set video etc.}
- T: transition function it describes the interaction of agents (system's architecture) to the environment (the probability of changing on the state to other state based on current state and the outcome of the applying action).
- R: reward shows the successor state due to the result of state and action interacting which are { high performance, least resource cost, video mode delivery}

The goal in reinforcement learning is to determine a policy a π(s) that maps each state to the action maximizing the total expected the future return of actions an in the state, s.

$$a^* \longleftarrow \text{argmax } Q(s,a) \text{ where } Q(s,a) = E[\sum_{k=0}^{\infty} r|s,a]$$

Equation 4.3 The best-so-far action calculation in RL

a* choose the best-so-far action for new planner operator.

4.3.2. Model-Based Reinforcement Learning

Model-based RL assumes knowledge of the transition matrix T, the reward function R, and the state and action space S and A which define the model of the world. This means that the expectation in Eq. 1 can be written explicitly in terms of T and R as Bellman equations (Bellman 1957):

$$Q(s,a) = \sum_{S'} T(s'|a,s) [R(s,a,s') + V(s')]$$

Equation 4.4 The next state and action in Reinforcement learning

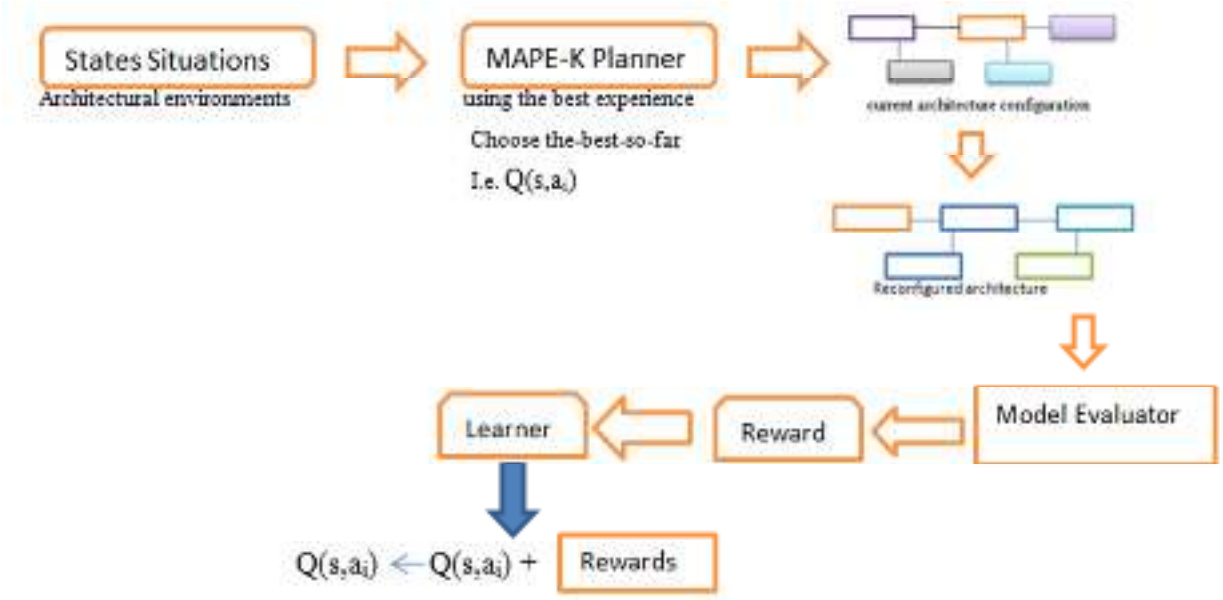


Figure 4.5. Transition process in MBRL

This research work uses model-based RL to model the system's architecture and realized RL. Model means a model or abstraction of the dynamics of the software systems environment. It enables a software model in which agent (software system) can learn from the environment and predict the outcome at the same time.

The reward in the self-adaptive system must be calculated in the form of the utility function. One of the necessary concepts that should be included is environmental '*situation'* which determines the system to know when it must detect and monitor the state and reward, and also when it must take the action corresponding to the state that results in the new state, which means architectural configuration. The situation also has an effect on choosing the available actions and serves as a triggering entity for monitoring and adaptation of the system. Because it can possible systems can stay on the different architectural configuration in similar states and actions. The existing reinforcement learning algorithm works to follow described in Algorithm 2.

Algorithm 2:

Input: the initial state of function agent

Output: best initial state and pi(s)= argmaxQ(s,a)

1. Begin
2. Loop:

   Give the initial state of function, Initialize Q(s, a) of function agent, gives the parameters alpha and γ.
3. Loop: Choose action a in state s by using e-greedy strategy
4. Loop(each step of each scene):
5. Choose the action a in the state s
6. Get the reward r and the next state s'.
7. Determines action a' in state s' by using e-greedy strategy
8. $Q_{t+1}(s, a) \leftarrow Q(s, a) + \text{alpha}\ [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(s, a)]$
9. Model(S, A) $\leftarrow$ R, S' (assuming deterministic environment)
10. s $\leftarrow$ s', a $\leftarrow$ a'
11. Until s is the terminated state.
12. Until all the Q(s, a) are convergent
13. Until all the initial state is tried.
14. End

As algorithm 2 states that, the (d) line shows the main learning algorithm which is called q-learning that are characterized by the model-free reinforcement learning algorithm.

Q(s, a) – store or accumulate q-values during the learning stage. It accumulates the value of the previous execution result based on the observed reward, current state and selected action as discussed in [35].

Alpha – step size parameter which is 0<alpha<1. Control the weight.

γ - controls the weights of action, in which $0<\gamma<1$. The action selection algorithm in line (b) is determined by the author as the behavior of action selection algorithm.

Model(S, A) – abstracts the whole environments of the self-adaptive systems.

For updating the model based on the rule of the policy improvement algorithm and to change the Q-learning algorithm to model-based reinforcement learning algorithm just we have to replace line (d) by using:

$$Q(s, a) \longleftarrow \sum_{s' \epsilon S,\, r \epsilon R} p(s', r|s, a)(r + \gamma V(s'))$$

Equation 4.5 Calculation on Updating model using MBRL

We can look the algorithm uses p(s', r|s, a), a probability defined by the MDP model, which uses a policy in the model-based algorithm.

Model-based RL can preserve a stable performance [25] by maintaining external control model that can predict the outcome before acting. As a result, training engineering knowledge about the essential system into a prior model toughly profits the learning development.

4.4. A Proposed Decision-making strategy for SAS

In reinforcement learning process during the selection of action and state, it uses either exploitation or exploration strategies. Thus the algorithm should balance the strategies in order to be effective as well as to speed-up the model updating process. Due to the learning parameter which is specified at design time in between 0 & 1 for representation uncertainty has related to exploitation and exploration strategies. If the learning parameter is an approach to 1, the learning strategy explores randomly the best action to maximize the returned reward which leads to performance delay due to searching space. As a result, the environment will be observable and deterministic. For that reason,

the issue of uncertainty will be neglected. In contrast the learning parameter in approach to 0, the algorithm exploits the best so far from previous experience. Due to multiple uncertainties the learning algorithm would be returned alternative action and reward and need to select the better one which intern delays the learning and updating model process.

Due to the above drawbacks of each algorithm on the learning process, the proposed solution adopted the next algorithm by combining algorithm 1 & 2. The merge of this algorism takes algorithm 1 first and follows the second on as shown in algorithm 3.

Algorithm 3:

Input: all utility values

Output: utilities of all strategy

1.Begin:

2. Loop given all utility values of each quality to each strategy, utility = 0

// switch to the highest utility with qualities to state and action pair

3. Do i to n times

4. Utility(Quality) ⟵ Utility(function + Impact)

5. Utility(Strategy) ⟵ utility + Utility(Quality) * preference

6. Until all utility of strategies has calculated

7. Until all utilities if qualities under each strategy has calculated

8. (s, a) ⟵ utility

9. $(s, a) \leftarrow (s', r)$ // Update the model

10. $p \leftarrow |r + \gamma \max_{a'} (s', a') - (s, a)|$

11. **Insert (s, a)** into Model, M with reward r

12. $Q(S, A) \longleftarrow Q(S, A) + \text{alpha } [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S, A)]$

13. Model(S, A) ⟵ R, S' (assuming deterministic environment)

Repeat n times:

S ⟵ random previously observed the state

A ⟵ random action was previously taken in S

R, S' ⟵ Model(S, A)

Q(S, A) ⟵ Q(S, A) + alpha[R + γ $\max_a$Q(S', a) – Q(S, A)]

14. End

The major issue in adopted algorithms is on how to map the result of the utilities' of each strategy to reinforcement learning problems in turn, again mapping reinforcement learning to self-adaptive system problems.

Due to utility outcome, the result will be stored as the initial values of the environment to the model. The algorithm also keeps the balance on the learning value due to the varying learning parameter from 0 to 1. Besides that, the adopted algorithm should not be obligated to use learning parameter=0.5 for balancing the exploitation and exploration strategy.

***Strategy Executor*:** this autonomic decision loop part performs adaptation action on the target system/managed system through effectors components. After that, the system will have a new architectural configuration or topology. Eg. addServer, removeServer, and set video and setImage in section 5.2 are the part of effectors in the implementation. Execute behavior is responsible for executing adaptation plans. Plans can contain multiple actions to be invoked to the managed system.

Actions are only executed when the managed system is ready for it is System ready to take action. Actions are executed sequentially by invoking an effector (invokeNextAction()). When the complete plan is executed, the behavior returns to wait for the next plan to be executed (PlanCompleted). As a pseudo code, it works as follows

```
Initial:  waiting for action

    Start: t<=period

          isSystemReadyToTakeAction()).

             Execute_RealseAction[Comp_ID]

           invokeNextAction()

          Execute_AddAction[Comp_ID]

       t.reset()
```

4.5. Model Construction Process

The model of the systems abstracts the whole environment infrastructure that enables the system to predict the future impact before it performs actions that leads to the architectural configuration. The

main work of in this model construction is how composites components reconfiguration reward parameters designed in different kinds of reinforcement leering. It encompasses problem domains, information (knowledge) about the systems that serve as a set of inputs for MAPE actions. The main activities in model construction processes that are described the whole procedures from requirement analysis to learning from the environments are: Discovering state, action, situations, Context analysis and uncertainty analysis, Action-uncertainty mapping, Drive state transition prior distribution are the basics all. The model component constitutes all architectural constraints, rules, regulations architectural model, previous experience, the reward of each configuration generally all information related to both managing system and managed systems. Each of the decision computing loops interacts with this component to justify the situation of the managed system as well as update them to manipulate accordingly to the changed environment. In the proposed framework, the utility theory performer components are embedded in the knowledge model part with learning table information.

A model predicts what the environment will do next.

$\mathbb{P}$ predicts the next states

$\mathbb{R}$ predicts the next (immediate) rewards e.g.

$$\mathcal{P}^{a}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
$$\mathcal{R}^{a}_{s} = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

Algorithm 4: updating the model

Inputs:

(s, a): State-action values, initialized arbitrarily

(s, a): Model, predicting the reward and successor state for taking action a in state s

s_0 : Initial state

Algorithm:

Initialize the model, M, to be empty

$s \leftarrow s_0$

Repeat

Choose a for the current state s based on **Q(s,a)**

Take action a, observe reward r and resulting state s'

(s, a) $\leftarrow (s', r)$ // Update the model

$p \leftarrow |r + \gamma \max_{a'} (s', a') - (s, a)|$

Insert (s, a) into Model, M with reward r

Repeat

$(s, a) \leftarrow$ M // Remove the least reward from the model

$(s', r) \leftarrow M(s, a)$

$(s, a) \leftarrow (s, a) + [r + \gamma \max_{a'} (s', a') - (s, a)]$

Compute priority for all states $(\bar{s}, \bar{a})$ that lead to s, and add to Model

Until all entries in M have reward r < r_0

$s \leftarrow s'$

Until the end of the learning episode

Return (s, a), the updated state-action values

The model construction process is designed based on the Markov Decision Process (MDP) that proved a solution to map the existing self-adaptive systems to reinforcement learning. Almost all reinforcement learning problems are described by MDP. As a result, the model construction process uses all concepts that exist in MDP as [1], [35] used it for their agent-environment interaction mapping problems. Thus the steps to construct the model as discussed below.

4.5.1. Discovering State, Action, Situations

Since the state is a key combination of conditions (important conditions) [9] that exists in the system, three different conditions are designed as requirements for state-action combination discovery. For modeling environments according to (Bellman 57) theory, the self-adaptive will describe by

- N states, K actions, discount factor $0 \leq \gamma \leq 1$; discount factors used for the representation of uncertainty.

For N states in some systems, there might be a number of states that classify the system systems in different ways such that;

S = {s1, s2, s3…sn} where the number of states which are a subset of S, as for an example Resource = {small, medium, large}, the system may exist with the utilization of a small number of resources, medium amount of resources or a large number of resources.

K actions also may be available that helps the system modifying its runtime components dynamically. A = {a1, a2, a3…an} represents set of actions the agent performed to informed other next state s_t. e.g. Action = {attach, detach, addResource, setVideo}

- step t, the agent informed state is s_t, chooses a_t
- receives payoff Rt- rewards; expected value is R(St, At)

After the system chooses an action and predicates the next state, it adds accumulated rewards in the knowledge model. In model-based reinforcement learning agent may get different rewards at similar state condition and action selection strategy. The common way to quantifying or calculating rewards function that is explained in most primary studies is distinguished accordingly the problem domain.

In other RL problems such that path navigation or robotics path, initial state and goal state will be identified in each movement that can be represented the whole state-action environment in the tree graph node. If an agent's current state has linked with goal state reward can be any positive number, if it has linked with the next state reward function is 0 else the reward function will be -1. All of these rewards express gain, loss or reward that have no effects on learning function. Rewards at a time in R(St, At) function for the proposed solution has multiple payoff reward function that depends on execution time, which is the goal of the system. Other goal states that have effects on reward functions are:

- Least resource consumed more preferred than large resource consumption.
- Video rather than an image for information delivery.

- the probability that next state is s' is T(St, At, S')

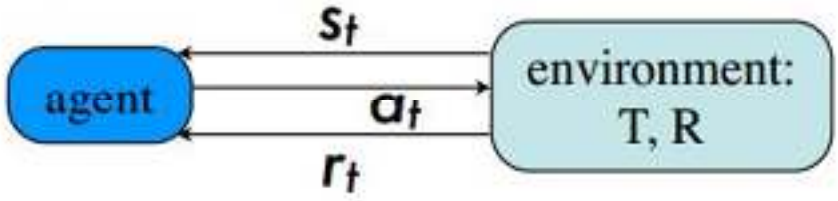


Figure 4.6 Agent and Environment interaction

$$Q(s,a) = R(s,a) + \gamma \sum_{set}(s,a,s') \max_{a'} Q(s',a')$$

Where γ is a discount factor:

Why discount factor? Uncertainty about the future may not be fully represented.
If the reward is financial, immediate rewards may earn more interest than delayed rewards.

The only thing left in MDP for mapping reinforcement learning problems to self-adaptive system environments is situations. Situations allow the agent (software systems) to control the states and rewards, for triggering the adaptation and observation activities in MAPE-K autonomic computing loop. Identifying the situation in one's system in its execution environment, it answers the questions [26], how to adapt, when to adapt and what to adapt. When the situation added on reinforcement learning problems of figure 4 in [17],

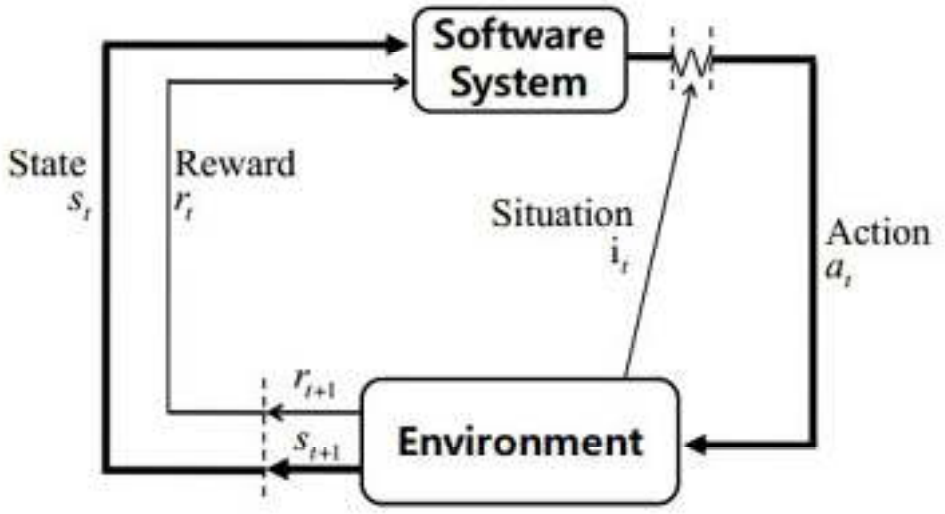


Figure 4.7 Agent and Environment interaction in the self-adaptive system

For reconfiguring the architecture topology there should have the system must change its state by specific action. Action leads to the system (agent) choose new state/possible configuration of architectural elements by attaching, detach connectors or add or remove components from configurations at a run time. The action would be the function or reconfigure process that leads the system to transform its state from one to another. [45]

The action in self-adaptive systems would be classified into two groups. The first group set of action that affects architectural configuration or topology. The second group is a set of action that affects functions that are providing as a service for the user, in other means, they are affecting source code

or methods. Like a reality, the above group of action has also related during reconfiguration in some ways.

4.5.2. Uncertainty Analysis

Most of the studies which have been issued problems and solution in self-adaptive systems, they ignore the underlying uncertainty and how they can be mitigated [19], within their proposed solutions. This proposed solution would describe uncertainties that are raised from the internal interaction of components and externally executing a process based on conditionally.

A. Uncertainty from manages the executor components that performs adaptation on the managed systems. Such kind of uncertainty arises due to the analytical model that represents complex-based systems. Systems which have composite components by design-time will have also a complex analytical model. Since the model will reason about the impact of adaptation action on different qualities of the system, will have some amount of percentage impact on the model. This quantification will make analyze by the utility theory that can be considered as a state for learning procedure.
B. Uncertainties from external source which is a load of CPU and Memory by external running software on specific machine are identified uncertainty, since the author can just imagine how much load exists on running machine or the allocation of enough space from self-adaptive system due to other will delay execution time in 2% (it is user defined).

4.5.3. Action-Uncertainty Mapping

As state-action mapping helps to solve the problems in reinforcement learning just, action uncertainty also important steps in model construction that helps to manage the uncertainty according to the framework stated in [64]. As based on the above kinds of uncertainty in A, calculate utility for the impact of multiple quality attributes action would be mapped and for uncertainty B: changing graphics mode to the image is the affected action for underlying uncertainty. But the uncertainty rose from inaccuracy of data or unreliable communication of architectural constraints needs managing of invariants among the components which can be solved according to the framework explained in [65] which is not issues in this study.

4.5.4. Drive State Transition

When a self-adaptive system needs to adapt from one to other states due to specific situations, there should be a state transition rule that performs reconfiguration accordingly. So that, for this activities transition state ways described in section 4.3.1 would be used. For better clarity, the whole part of this step is discussed with the prototype that can use the proposed solutions.

4.6. Proposed Model-Checking for Stochastic Behavior Analysis

For assurance and the verification of the model after reconfiguration calculating the impact of the action on the model is vital action. As a result, the proposed solution is discussed Discrete-Time Markov Chains for model checking part [45].

Definition: A DTMC is fully probabilistic; it is given as a (finite) set of states' S, a subset of initial states $\bar{S} \subseteq S$ and a transition probability matrix P: S x S [0,1]. For each pair s, s' of states, the probability of making a transition from s to s' is given by P(s,s'). we require that $\sum_{s' \epsilon S} P(s,s') = 1$ for all states $s \epsilon$ S. terminating states can be modeled by adding a self-loop, i.e. setting P(s,s) = 1.

Examples of impact model for reconfiguration action swichToImageMode is

```
1 define S=(s:Server | !s.isImageMode and s.isActive) and define n=size(S)
2 define f(x)=x*(1-n/(2*(n+1))) and define g(x) =x*(1-n/(n+1))
3 impactmodel switchToTextualMode
4  n>0 -> {[0.7] {forall s:S | s.isImageMode'=true & forall c.VideoMode | c.expExeTime'=f(c.expExeTime)}
          +[0.3] {forall s:S | s.isImageMode'=true & for all c. VideoMode | c.expExeTime'=g(c.expExeTime)}
```

This model explains the impact of the reconfiguration action over the system's architectural properties, setImage() function where the property isImageMode to true is denoted by s.isImageMode'=true. Additionally, the model foresees that switchtoImage can impact the execution time of the media player.

Alternatively, switchToImageMode can impact the execution time of media player calling function,

```
1 impactmodel switchToTextualMode
2 n>0 ⟶  forall m:MediaP | {[0.7] {forall s:S | s.isImageMode'=true &
  forall c. VideoMode | c.expExeTime'=f(c.expExeTime)}
3          +[0.3] {forall s:S | s.isImageMode'=true & for all c. VideoMode | c.expExeTime'=g(c.expExeTime)}
```

While the model deliberated that the property isImageMode of servers is issue only to system control, isActive is defined as monitored property and it was considered that the activation of server, through the execution of addServer functions. The aspect of this model impact is presented below.

```
1 define n=size(s:Server | !s.isActive) and define S=(s:Server | s.isActive)
2 define f(x)=x*(1-((1/log(100*n,2))*(n/(2*n+1)))) and define g(x) =x*(1-1/log(100*n,2))
3 impactmodel addServer
4  n>0 → {[0.90] {foreach s:S | s.isActive''=true &
5                          {[0.7] forall  c. VideoMode | c.expExeTime'=f(c.expExeTime)
6                          + [0.3] forall c. VideoMode | c.expExeTime'=g(c.expExeTime)}}
7                             + [0.10] {forall s:Server | s.isActive'=s.isActive & forall c. VideoMode |
c.expExeTime'=c.expExeTime}}
```

This impact model indirectly states that the running of the server is estimated to be unsuccessful with probability 0.10 and foresees that the adaptation action may impact client response time in two ways, both considering the number of servers that were already active. Moreover, isActive is also defined as a monitored property, since a server could become spontaneously inactive (e.g., due to a server crash). The impact model above does not define any impact of the adaptation action over the property isActive of already active servers, hence the probability of an active server crashing while executing addServer is considered to be so small that it can be neglected. Alternatively, we can define the probability of each relevant crash scenario (e.g., for one server, two servers, three servers). For instance, the impact model presented below defines that the probability of exactly one active server crashing while executing addServer is 0.001.

```
1 define T=(s:Server |s.isActive)
2 impactmodel addServer
3 m>0 → {[0.999] {foreach s:S | s.isActive''=true &
5                          {[0.7] forall  c. VideoMode | c.expExeTime'=f(c.expExeTime)
6                          + [0.3] forall c. VideoMode | c.expExeTime'=g(c.expExeTime)}}
7          + [0.001] { foreach s:S| s.isActive'=true & foreach t:T | t.isActive'=false &
                    forall c: VideoMode | c.expExeTime'=c.expExeTime}}
```

CHAPTER FIVE

5. Implementation of the Proposed Solution

5.1. Overviews

As described earlier in chapter two, Znn.com website has adopted some data that are required to utility outcome calculation for decision-making purposes during the adaptation period. These data is specified at design time according to the interest of owners and developers. Currently, the development of a self-adaptive system which has a capability to reconfigure dynamically uses machine learning strategies to mitigate unpredictable circumstances. Such a paradigm also included in systems like the real-time embedded system, robotics, and big data analytic systems. The implementation demonstrates how to design and implement a simple desktop-based application that can reconfigure its architectural elements by learning its operating environments dynamically which illustrates further the DSARCC framework working principles. Additionally how to code (feasible) each of components exists in the framework and also how to interact before and after adaptation will be discussed with the corresponding implementation.

5.1.1. Working Environments

- Laptop
    - Processor: Intel® Core™ i3-5005U CPU @ 2.00GHz 2.0 GHz
    - Installed memory (RAM): 4.00GB
    - System type: 64 –bit operating system, x64-based processor
- IntelliJ IDEA: 2018.1.3 (community edition): it uses as scripting editors for both java and python programming languages. Based on it simplifies the way and process of integrating two different tools. So it is better writes java and python language on a single tool rather than writing separately on different tools. Such processes have an impact on the adaptation process of execution time. The followings are runtime execution environments exist in IntelliJ IDEA.

- JRE: 1.8.0_152-1136-b38 and 64
- JVM: OpenJDK 64-Bit Server
- Python interpreter: python 3.6/Anaconda3/python.exe

5.1.2. Programming languages:

- Java – for user interface, component, and connector implementation which are the major constitutes of software's architecture of one system. Besides that, it serves for the development of utility outcomes for static learning capability.
- Python – for implementation of reinforcement learning algorithm and learning strategies because it has better numerical and scientific libraries that provides an environment of analysis of data in terms of number.

  In the next part, the prototype of the self-adaptation system would describe briefly according to the framework. The system is a simple desktop application that provides information for users either in image graphics or video graphics. The software consists of three basic components which have internal components as a composite component structure. These are server component, information displaying component, and main user interface components. The high-level architecture of the prototype is shown in figure 5.1

5.2. Framework Mapping in the Prototype

Generic Architecture/High-level architecture of the prototype application is considered the external controller in DSARCC framework. In the next section, the element of the generic architecture will be described in the low-level design of architecture with their corresponding components of the framework that are mapped to the prototype. This generic architecture shows the proposed solution strategies by combines the utility theory concept and a reinforcement learning approach. The architecture is based on layered fashion architecture, which is three-tier architecture with translation infrastructure. These three layers are Goal management, Change management layer and component control layer which described detail with their components in. In this work, the management system behavior, as well as the capability of learning, is new addition components to the work.

The next figure shows the interaction of managing components to managed systems through components interface. The managed system consists of three main parts server, functional components and User Interface. The servers store graphical information that displayed either in image or video formats data and provides information about Tourism. At initialization time, only one server becomes active. After a while when more clients access the system and system's response time becomes above the threshold, other servers become active. This action will be reduced response time but it needs more resources. In contrast the number of clients that are

accessed the system becomes decrease, some servers will be deactivated due to managing the executing resource in the least cost.

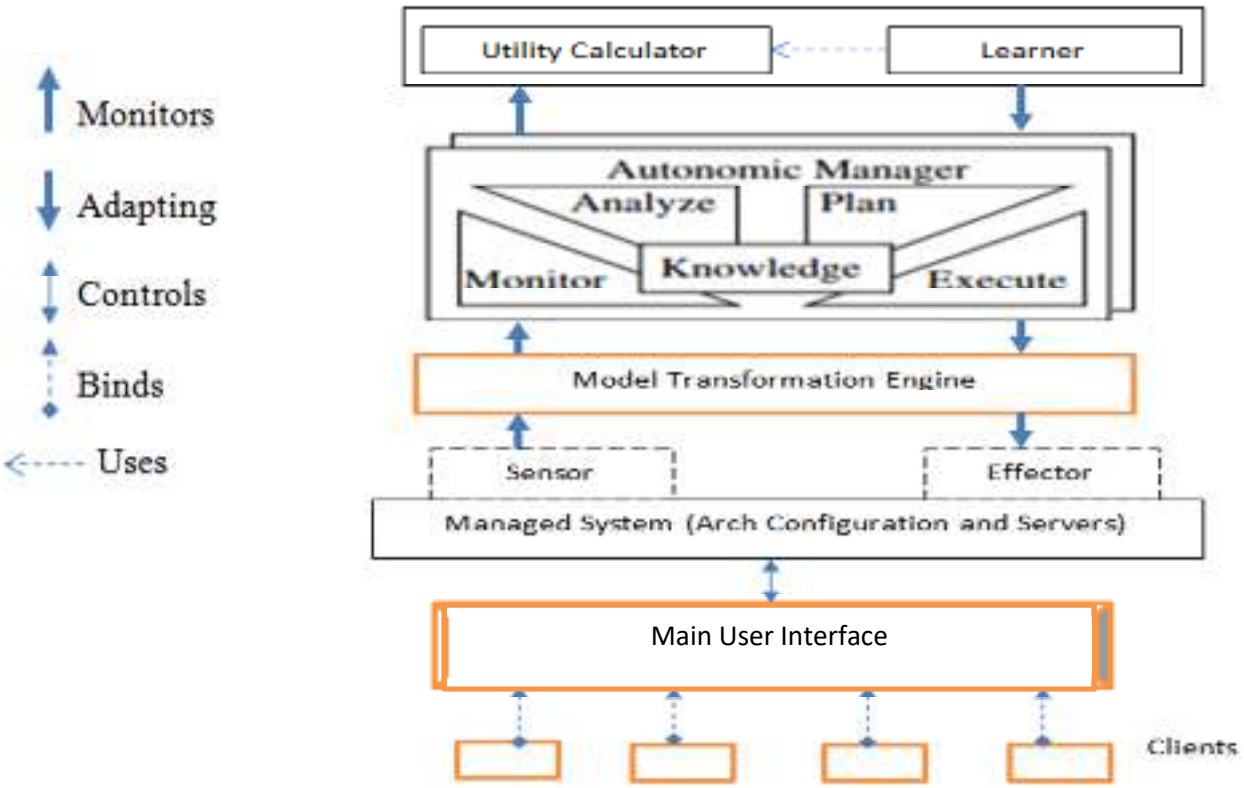


Figure 5.1 High level generic architecure of the prototype

5.2.1. Execution Environments

It consists of the details runtime information from the managed resources e.g. topology information, metrics (e.g. offered execution time, memory and CPU consumption), configuration property settings and so on. The states of the environment that are observable at runtime collectively called state space.

The state space of software includes:

Information delivery = {ID[Video, Image]}

System Resources = {SR[Small, Medium, Large]}

Execution time = {RT [Normal, Low]}

Based on the above states, state space will be combined through combination from each set of state. As a result, the system would have 2x3x2=12 different possible states as shown in figure 5.2. These state spaces are:

{[Video, SRSmall, RTNormal], [Video, SRSmall, RTLow], [Video, SRMedium, RTNormal], [Video, SRMedium, RTLow], [Video, SRLarge, RTNormal], [Video, SRLarge, RTLow], {[Image, SRSmall, RTNormal], [Image, SRSmall, RTLow], [Image, SRMedium, RTNormal], [Image, SRMedium, RTLow], [Image, SRLarge, RTNormal], [Image, SRLarge, RTLow],

Add/down resource ⟶

Switch to video/Image - - - - - - - - - - ⟶

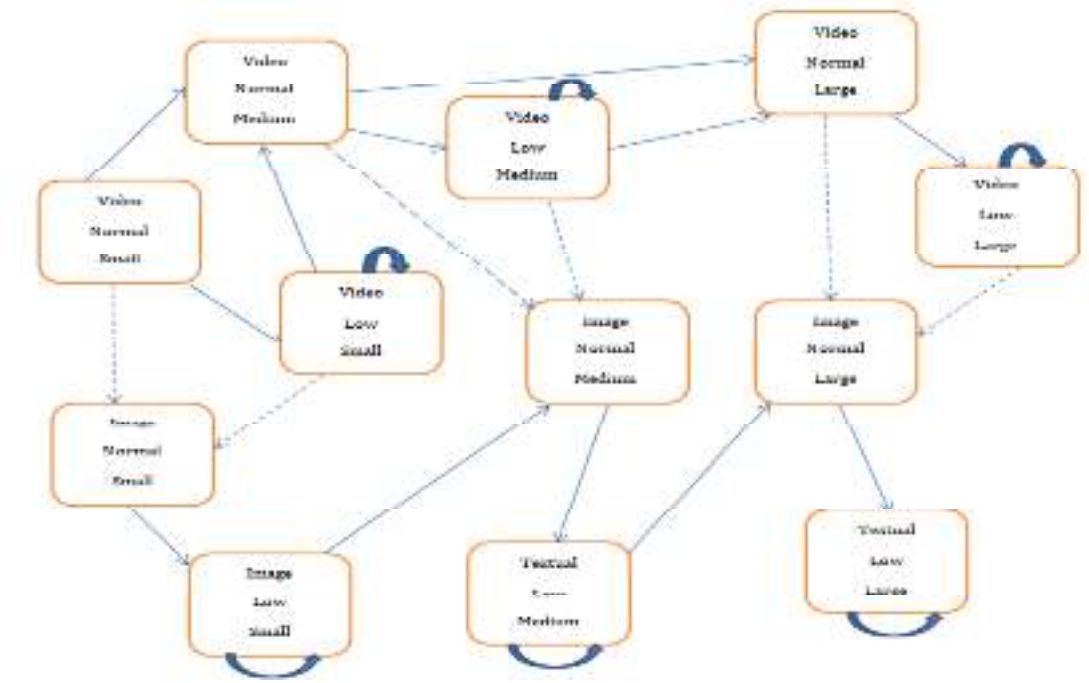


Figure 5.2 State and action representation in the environment

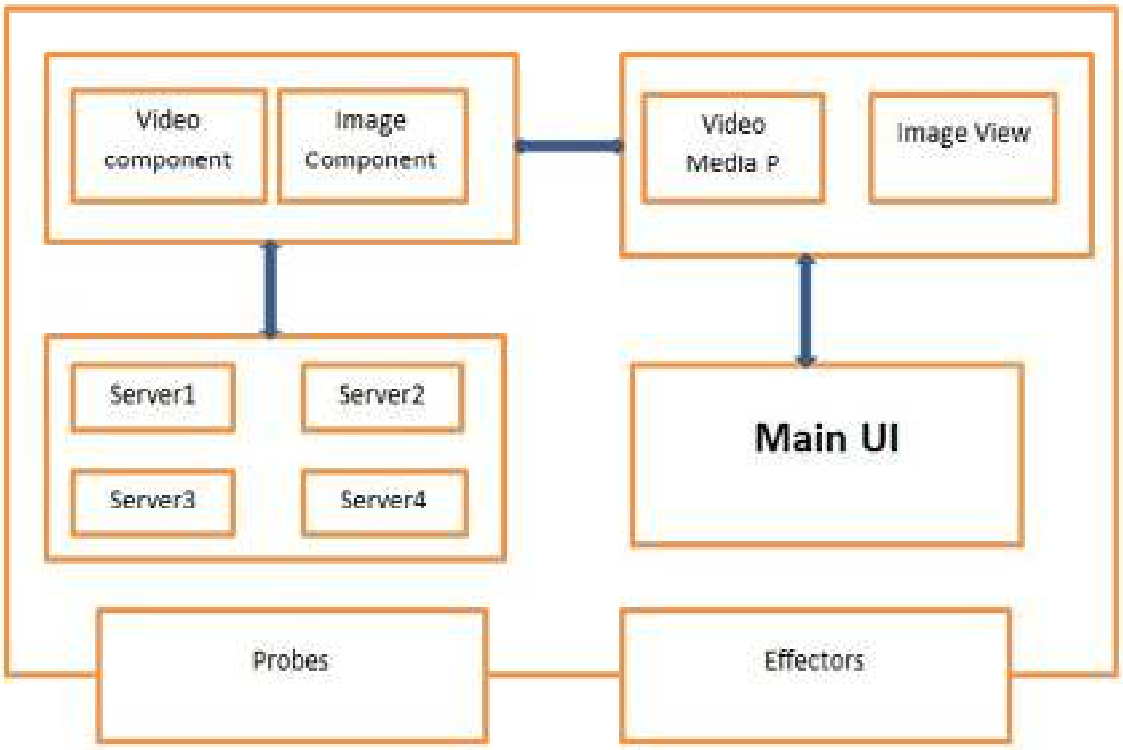


Figure 5.3 Main user interface for the prototype

As the figure 5.3 shows that each component has multiple components inside there which can be executed at runtime or remove those from being operated with their connector based on the operation environment according to architectural constraints specified at analysis part of MAPE-K loop. The probes and effectors components use for gathering the runtime information for feedback managing loop and perform the reconfiguration action results according to the model checking component results on managed software respectively. The major activity in this developing of the prototype is to separate the functional part of the software from the Probes and Effector components.

The next part of the development is developing the managing component that controls the runtime execution information and makes a decision whether the software is being adapted or not. This part will be implemented in a component class fashion that takes different parameters which describes the architectural quality of the system and the runtime collected metrics. The next figure shows high-level architecture that shows the MAPE-K feedback loop with utility performer and learning action components.

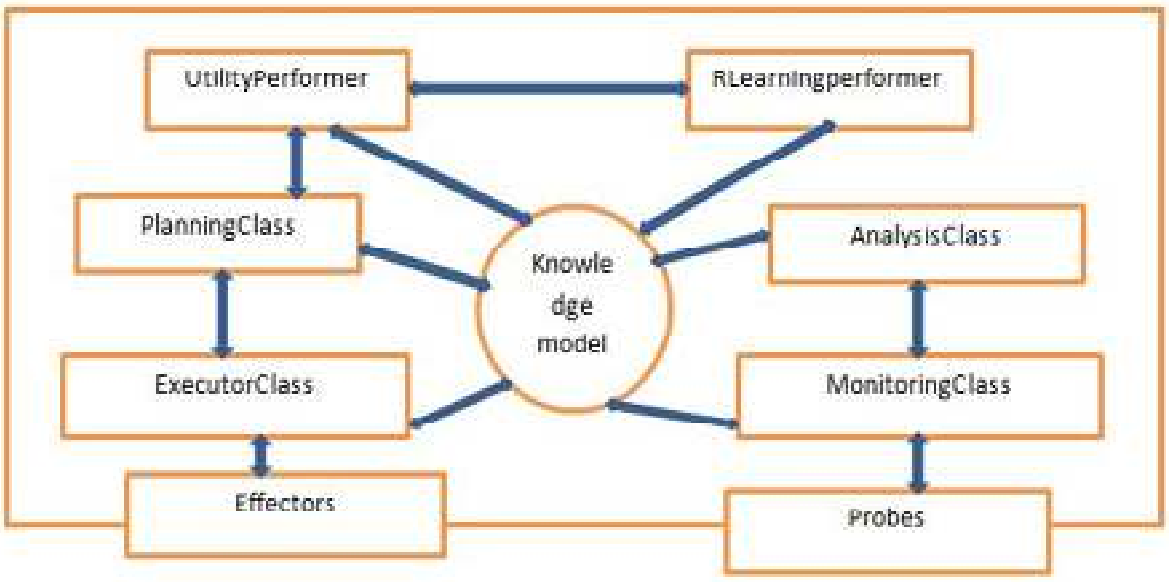


Figure 5.4 High level architecture of managing system

- As figure 5.2 shows all classes or components to feasible the managing parts of DRSACC frameworks. In this architectural description of how this research work implements the whole self-adaptive software system is briefly stated in class, operation, and parameters of the components' that served as a connector of the architectural components. Below, there is a short brief explanation of each class which is in Appendix 2 and 3.

5.2.2. Adaptation inputs:

Utility Function

From table 5.1, it can be inferred that, the utility is higher for execution time values close to 100% and lower values are heavily punished because of their utility function. In addition, the defined values lead to low consumption of computational resources by designating one active server as the best utility outcome of the system. With regard to the experienced load, the system favors a low utilization capacity.

Table 5.1Utility of each strategy

Execution time		***Active servers***		***System's load***		***Mode utility***	
Value	utility	Value	utility	Value	utility	Value	utility
20	1.0	0	0	100	0.0	2	1.0
30	0.9	1	1.0	90	0.05	1.7	0.9
40	0.7	2	0.85	80	0.1	1.5	0.75
45	0.5	3	0.55	70	0.4	1	0.5
55	0.3	4	0.3	60	0.5	0.7	0.3
60	0.1	>4	0.09	50	0.6	0.5	0.25
70	0.05			40	0.7		
80	0.03			30	0.8		
90	0.002			20	0.9		
>90	0.001			10	0.95		

Static impacts on the quality dimensions for strategy

In Table -- the adopted values in our case-study and it should be noted that the addServer strategy increases both availability and used computational resources, at the same time it reduces the system utilization since it has more machines to process the same demand of requests. Conversely, both down server strategies will reduce costs and increase the use of the system. When deciding to discharge one server, the system will increase availability.

Table 5.2: Static impact of quality dimension

	Performance	***Cost***	***utilization***	***Mode***
Add extra server a server	+15	+1.0	-20%	1
Down server	+5%	-1.0	+20%	0.7
Switch to video mode	20%	+1.0	-30%	1
Switch to Image mode	+0%	-1.0	+10%	0.5

- **Utility preferences**

The preferences are shown in Table 3 and it can be noticed that execution time is twice as important as the cost or utilization of the system. Although they may not be achieved optimally, these preferences can be used to solve resource constraints or trade-offs between certain quality attributes. It can be edited as a dynamic impact that can be changed at runtime or can be specified by the

interest of the developer at design time. Because of that, the reconfiguration result must keep the preference of each quality attributes. The utility preference of each adaptation strategies with values are:

Table 5.3 Utility preferences of quality goals

	Percentage(%)
Execution time	45
Utilization	20
cost	20
Mode quality	15
Total	100

The above adaptations inputs can be prepared either are adopted from the research work [7] or can be designed as based on utility priorities raised from the owner of the software according to input and self-adaptation environment design phases [9].

In a self-adaptive system, the main activities are occurring in the planning components because it chooses new state and action that are compatible with the current environment to maintain the architectural constraints. So that utility income and the learning part is the heart of this action. Next how the utility outcome is calculated will be shown.

At a specific time, the monitoring class collects the following runtime metrics, the utility outcome of each strategy is calculated below based on Utility theory in chapter 4.

Number of servers running = 2

Executing time = 7s

exe.time = ((Executing time-5)*100)/5=40%

Mode of delivery = Image

Load = 80%

Add Server:

UPER = Utility(exe.time + impact) = Utility(55) = 0.3

UARES = Utility(No. of servers + impact) = Utility(3) = 0.55

ULOAD = Utility(Load + impact) = Utility(60) = 0.5

UMODE = Utility(Mode + impact) = Utility(1) = 0.5

Utility = UPER * Performance preference

+ UARES * Cost preference

+ ULOAD * Utilization preference

+ UMODE * Mode preference

= 0.3*45% + 0.55*20% + 0.5*20% + 0.5*15%

= 42%

Down Server:

U_{PER} = Utility(exe.time + impact) = Utility(45) = 0.5

U_{ARES} = Utility(No. of servers + impact) = Utility(1) = 1

U_{LOAD} = Utility(Load + impact) = Utility(100) = 0

U_{MODE} = Utility(Mode + impact) = Utility(0.7) = 0.3

Utility = U_{PER} * Performance preference

+ U_{ARES} * Cost preference

+ U_{LOAD} * Utilization preference

+ U_{MODE} * Mode preference

= 0.5*45% + 1*20% + 0*20% + 0.3*15%

= 67%

Switch to Video:

U_{PER} = Utility(exe.time + impact) = Utility(60) = 0.1

U_{ARES} = Utility(No. of servers + impact) = Utility(3) = 0.55

U_{LOAD} = Utility(Load + impact) = Utility(50) = 0.6

U_{MODE} = Utility(Mode + impact) = Utility(1) = 0.5

Utility = U_{PER} * Performance preference

+ U_{ARES} * Cost preference

+ U_{LOAD} * Utilization preference

+ U_{MODE} * Mode preference

= 0.1*45% + 0.55*20% + 0.6*20% + 0.5*15%

= 35%

Switch to Image:

U_{PER} = Utility(exe.time + impact) = Utility(40) = 0.7

U_{ARES} = Utility(No. of servers + impact) = Utility(1) = 1

U_{LOAD} = Utility(Load + impact) = Utility(90) = 0.05

U_{MODE} = Utility(Mode + impact) = Utility(0.5) = 0.25

Utility = U_{PER} * Performance preference

+ U_{ARES} * Cost preference

+ U_{LOAD} * Utilization preference

+ U_{MODE} * Mode preference

= 0.7*45% + 1*20% + 0.05*20% + 0.25*15%

= 56.25%

The above calculation shows that if the system down one server, the reconfiguration would result in better utility to keep the architectural constraints. After reconfiguration, the system collects its execution time and the system's load with the mode and the new active servers to calculate the new utility outcome.

For learning purpose, the prototype is demonstrated by combining two steps. The major activity in this part is preparing environment and agents that can represent the prototype as well as its execution environments. So it needs a careful mapping of the software environment entities as a state, action transition probabilities, and rewards to make reinforcement learning possibilities based on MDP.

Due to that, the proposed work implements the learning phase of the prototype in two ways. The first one is using [n]x[m] matrix representation of state and action by identifying the states and the applying action as proposed in chapter 4, the representation is feasible by classifying the collected metrics in 12 possible ways. Then by making a tree on the basis of Discrete-time Markov Chain process by distinguishing initial state, goal state, and the path to the goal state. These three parameters are collected at runtime during the software is operated and passed to the python learning class for preparation of matrix representation. Concurrently the effect of action and the intended action at each state also submitted to the method that defines available state and action. Hereby the procedure will be discussed in the steps for a better explanation by beginning the first ways.

Here the first three lines are codes that take the collected metrics from the Java language.

Step2: Based on the above states and the information figure 6 shows, in chapter four the matrix gives values:

- 0 – if there is link or path among states
- 10 – if there is link or path among states and goal state (it can be any number other 0 and -1)
- -1 – if links or path does not available among states and generates matrix representation.

For example from the above states, the goal states calculated from utility outcome is 4 the generated matrix representation of the current environment would be like:

```
goal = goal state
initial_state = initialState;

R = np.matrix([ [-1,10,0,10,-1,0,-1,10,-1,-1,-1,-1],
                [10,-1,-1,-1,-1,0,10,-1,-1,-1,-1,-1],
                [10,-1,-1,10,-1,-1,-1,10,-1,-1,-1,-1],
                [10,-1,0,-1,0,-1,-1,-1,10,-1,-1,-1],
                [-1,-1,-1,10,-1,-1,-1,-1,10,-1,-1,-1],
                [10,10,-1,-1,-1,-1,10,-1,-1,-1,-1,-1],
                [-1,10,-1,-1,-1,0,-1,-1,-1,-1,-1,0],
                [10,-1,0,-1,-1,-1,-1,-1,-1,-1,0,0],
                [-1,-1,-1,10,0,-1,-1,-1,-1,0,0,-1],
                [-1,-1,-1,-1,-1,-1,-1,-1,10,-1,-1,-1],
                [-1,-1,-1,-1,-1,-1,-1,10,10,-1,-1,-1],
                [-1,-1,-1,-1,-1,-1,10,10,-1,-1,-1,-1]])
```

Q = np.matrix(np.zeros([12,12])) // at initial stage the matrix is filled by all zero's until the system gathers information and represented the environment accordingly.

Set uncertainty values, learning parameters with training episodes. In the case, this one procedure the uncertainty values are represented a number of running process in the machine and the internal structure (composite structure) of the software. If the uncertainty value is near to 1 the agent (software) tries to perform random action selection strategy, in contrast, if it nears to 0 it exploits the best so far experiences.

```
N_STATES = 12 # possible states exist in the grid
N_EPISODES = 25 # number of episode for starting the learning decision making
```

MIN_ALPHA = 0.02 # constant step-size parameters

gamma = 0.8 # discount factors

eps = 0.2 # uncertainty representing

q_table = dict() # model for store state, action in each learning phase

Table 5.4 Comparison terms for result and evaluation

Comparison terms	***Parameters***
1. Utility outcome based reconfiguration Vs Reinforcement learning based reconfiguration Vs Combination-based Reconfiguration	Similarities of predicate state and action with actual state and action in 10 situations
2. Utility outcome with MFRL Vs Utility outcome with MBRL	Reconfiguration results of state and actions goal/reward(qualities) Vs the utility preferences
3. The exploitation Vs exploration with the exploit-explore tradeoff	By changing the learning parameters, e=0, e=1, e=0.5

Generally, this chapter discussed the implementation of Information delivery self-adaptive system (IDS) by different means that helps for comparing the results in the evaluation and discussion part. As a result, the next chapter, Evaluation, Result, and discussion will discuss the results of the table 5.4 comparison evaluations.

CHAPTER SIX

6. Evaluation, Result, and Discussions

6.1. Results and Discussion

In evaluation part, the methodology and the internal structures, algorithms, and prototype is justified and done accordingly *software Engineering for Self-Adaptive Systems: research roadmaps, 2013* procedures. So that, the solution framework and prototype issued the feedback organization, the target resources for reconfiguration, and the reason for adaptation are in this study. Learning algorithms, model representation, which answers the question how dynamic learning is introduced into self-adaptive systems, are discussed in the proposed solution. Finally assurance way, handling of exploit-explore trade-off, adaptation control unit and reconfiguration pattern are also justified in the solutions.

For assurance and validation, testing and model checking are used [10] to evaluate and discuss the results of the developed prototype. For results purpose, 10 different state parameters are collected during the reconfiguring of the system at execution time. The results are:

6.1.1. Collected Runtime Metrics

Time 1:

- Num. of server = 1→ Small
- Execution time = 5→ Normal
- Mode = Video→ V

State → 2

Time 2:

- Num. of server = 2→ Medium
- Execution time = 5→ Normal
- Mode = Video→ V

State → 1

Time 3:

- Num. of server = 2→ Small
- Execution time = 14→ Low
- Mode = Image → I

State → 11

Time 4:

- Num. of server = 1→ Small
- Execution time = 4→ Normal
- Mode = Image→ I

State → 7

Time 5:

- Num. of server = 2→ Medium
- Execution time = 5→ Normal
- Mode = Image → I

State → 8

Time 6:

- Num. of server = 1→ Small
- Execution time = 6→ Low
- Mode = Image → I

State → 12

Time 7:

- Num. of server = 2→Medium
- Execution time = 5→ Normal
- Mode = Video→ V

State → 1

Time 8:

- Num. of server = 2→ Small
- Execution time = 8→ Low
- Mode = Image → I

State → 11

Time 9:

- Num. of server = 1→ Small
- Execution time = 5→ Normal
- Mode = Image → I

State → 2

Time 10:

- Num. of server = 2→ Small
- Execution time = 5→ Normal
- Mode = Video→ Video

State → 1

Based on the above-collected runtime metrics the results of the comparison in table 5.4 calculated against the actual reconfiguration (new state and action) with the predicate changes in state action scenario (state-action space explosion) in figure 5.2

The predicate state is the set of all states in which one state has path or link to them and the predicate action is a set of all actions that can be performed on the state as shown in table 1 & table 2 respectively.

Table 6.1Set of predicated actions

State	***Predicate actions***
State1	{setImage, downS, addS}
State2	{setI, addS}
State3	{setI, addS, downS}
State4	{setI, downS}
State5	{setI, downS}
State6	{setI, addS}
State7	{setVideo, addS}
State8	{setVideo, addS, downS}
State9	{setV, downS
State10	{setV, downS}
State11	{setV, addS, downS}
State12	{setV, addS}

Table 6.1 shows that possible set of actions in predicate actions columns which are applied in corresponding states in Column State and Table 6.2 shows the set of possible states in Predicate states that the self-adaptive system reconfigured itself to other state.

Table 6.2 Set of predicated states

State	*Predicate states*
State1	{2,3,4,6,8}
State2	{1, 6, 7}
State3	{1,4,8}
State4	{1,3,5,9}
State5	{4,9}
State6	{1,2,7}
State7	{2, 6 , 12}
State8	{1,3,11,12}
State9	{4, 5, 10, 11}
State10	{9}
State11	{8,9}
State12	{7,8}

6.1.2. Actual Metrics after Reconfiguration

The next table shows the triggered action (actual action) that is taken when the system needs to perform reconfiguration in UB, MFRLB, MBRLB, UB+MFRLB, and UB+MBRLB reconfiguration ways on the same collected runtime metrics.

Table 6.3 Set of actual new state and action in all adaptation type

ADN→NS	*UB*	*MFRLB*	*MBRLB*	*UB + MFRLB*	*UB + MBRLB*
	(NA→NS)	(NA→NS)	(NA→NS)	(NA→NS)	(NA→NS)
1→S2	addS→S1	setI→S10	addS→S1	addS→S10	setI→S7
2→S1	setImage→S8	addS→S1	downS→S9	addS→S9	addS→S8
3→S11	downS→S12	setI,addS→S9	setV,addS→S6	addS→S9	downS→S11
4→S7	addS→S8	downS→S7	downS→S6	downS→S2	setV→S6
5→S8	downS→12	setI→S1	setI→S6	downS→S10	addS→S1
6→S12	addS,setV→S7	addS→S8	setI→S6	addS→S12	addS→S8
7→S1	setI→S8	addS→11	addS→S4	downS→S8	downS→S2
8→S11	downS→S12	setV→S4	downS→S9	setV→S7	addS,setI→S9
9→S7	addS,setV→S1	addS→S10	setV→S11	setV→S2	downS→S6
10→S1	setI,downS→S6	setV→S1	addS→S4	setI→S8	addS→S4

Based on table 6.3, which specify the actual state and action and table 1&2 that specify the predicate state and action, the transition probabilities of reconfiguration ways out of 10 collected runtime metrics which have similarities is shown in table 6.4:

Table 6.4 Probabilities of the transition state in each reconfiguration ways

Reconfiguration ways	***Actual action over predicate action***		***Actual state over predicate state***	
	Yes	No	Yes	No
Utility based	9/10	1/10	5/10	5/10
MFRL based	9/10	1/10	3/10	7/10
MBRL based	9/10	1/10	7/10	3/10
UB+MFRL	9/10	1/10	6/10	4/10
UB+MBRL	10/10	0/10	9/10	1/10

Table 6.4 implies that almost 90% of the predicated set of action is feasible in actual triggered actions. Searching and exploring the required action from predicate is more or less similar across all kinds of reconfiguration ways. However, when utility based output is combined with model-based reinforcement learning, the planning component can searches all actions from the predicate set of action. On the actual state over predicate state side, the newly discovered state in utility-based has 50% similarity with predicated set of states, 30% in model-free reinforcement learning because of its focuses on only the rewards in the current state and action as works in [17], in contrast, it can have better ability to explore additional states when it combines to utility based reconfiguration ways because of the UB is considering some of uncertainty during runtime reconfiguration. The state space exploration has more similarities in model-based reinforcement learning as clearly showed the MBRL has an ability to learn the environment as well as it can also predict the impact of its configuration that enables it to manage uncertainties. But no ways can have similarities fully 100% with the predicate, because of that it's impossible to model the uncertainty that can happen at runtime perfectly.

6.2. Model-checking

In this ways of validation and assurance of self-adaptive software system, as shown in the implementation part, before reconfiguration occurred the model has been checked whether the new reconfiguration obeys the architectural constraints. Therefore, among reconfiguration ways, the result with high model checking utilities would have better learning and reconfiguration capabilities.

To perform such action, the system collects execution time of each reconfiguration action by model checking component which is the main goal of (reward) to the adaptation process.

Table 6.5 New quality goals due to reconfiguration ways

ADN	***UB***	***MFRL***	***MBRL***	***UB+MFRL***	***UB+MBRL***
1	V, N, M	V, N, M	V, N, M	V, N, LA	V, N, L
2	I, N, M	V, N, LA	I, N, S	I, L, LA	I, N, S
3	I, L, S	I, N, M	V, L, M	V, L, M	V, S, M
4	I, N, S	I, N, S	I, N, S	V, N, S	I, N, M
5	I, N, S	I, N, M	V, N, S	I, N, M	V, N, M
6	V, N, LA	V, L, M	V, N, S	V, N, M	V, L, L
7	I, N, M	V, N, L	V, N, LA	I, N, M	I, N, S
8	I, L, S	V, L, S	V, L, LA	I, N, S	V, N, S
9	V, N, LA	I, N, M	I, N, S	V, N, S	I, N, S
10	V, L, S	V, N, S	V, N, S	I, L, S	V, N, 1
Aggregation	**4V, 6I** **7N, 3L** **5S, 3M, 2LA**	**6V, 4I** **8N, 2L** **3S, 5M, 2LA**	**7V, 3I** **8N, 2L** **5S, 2M, 3LA**	**5V, 5I** **8N, 2L** **4S, 4M, 2LA**	**6V, 4I** **9N, 1L** **5S, 3M, 2LA**

V=video, N=Normal, I=image

As the above table shows that it helps to compare the model checking result against the specified quality attribute preferences of table 5.3. In aggregation row of the table, all reconfiguration ways are similar in the preservation of execution time (which has 45% preferences) although UB+MBRL have kept in more percentage in which 90% of reconfiguration did it. UB based registered low percentage; the other else are exhibited an equal percentage. In the case of others qualities which affects the earned reward in total aggregation, utility-based denies the priority of video over images, model-free RL denies preferred of less server using than large server, utility-based MFRL equals the relative impacts of Video and Image as well as using less resource with using more resources, finally both MBRL and utility-based MBRL keeps the architectural constraints specified in all quality attributes.

The existing methodology which is ABSA has different effectiveness with only utility theory, learning ability and combination of both utility and learning ability. As a result, the comparisons on the existing framework against the proposed one are visible when we evaluate the aggregation of

each reconfiguration ways on the table 6.5 UB+MBRL is conserved all utility preferences for each of adaptation strategies, the others are missing the utility preferences rule.

On the other side, the proposed framework also exhibited better performance in which the reconfiguration results yield normal execution time in which 90% against the existing and the others one. Finally, in the result part, it possible to discuss the number of episodes against scores as [17], enables to compare the exploitation-exploration tradeoff in the generated graph from python code.

Figure 6.1 shows the system uses a random selection of an action for next state, it can't learn from the environment unless it designs online planning, in contrast, figure 6.2 shows the system exploits

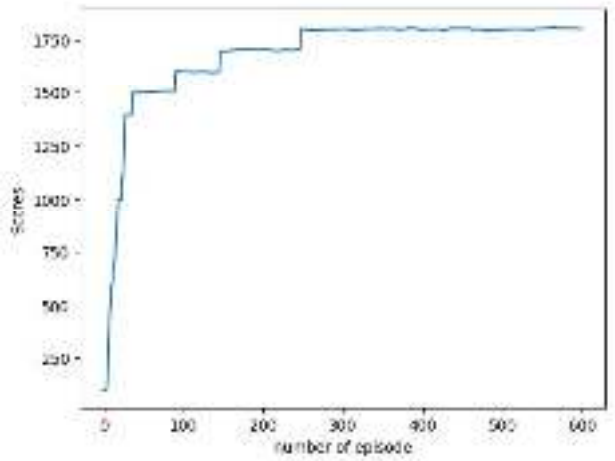


Figure 6.1 Exploration strategy

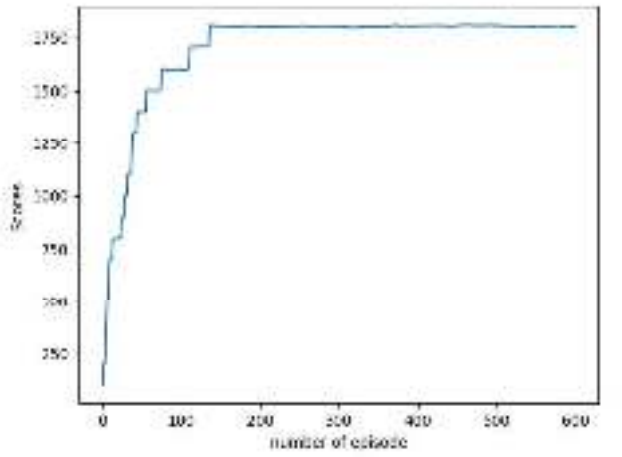


Figure 6.2 Exploitation strategy

the previous best-so-far configuration for the current state, it can also learn from the environments. At last, figure 6.3 shows the trade-off between exploitation and exploration, in which the system starts to exploit from previous experiences that stored in the knowledge model. If the reward is not best, immediately it tries to select actions randomly. So the figure also shows, the value of epsilon maintains the balance of between exploit-explore strategy. As the result shows, using utility theory principle would decrease action selection strategy and provides more so-far experiences. Therefore, the tradeoff between explore-exploit can be managed.

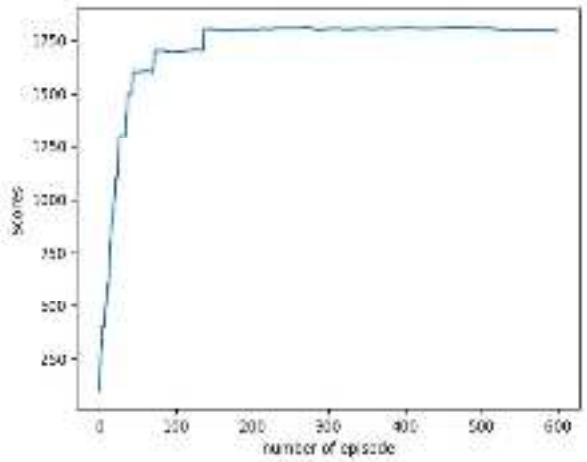


Figure 6.1 Exploration-exploitation tradeoff

6.3. SAFU Evaluation result

Finally, the proposed solution framework evaluated with the existing framework called Rainbow on the basis of SAFU which is a common quantitative metric.

Table 6.6 SAFU evaluation result of the framework

Criteria	***Value***	***Weight***
- Execution time	90	45
- Mode of delivery	60	15
- Cost	80	20
- Resource utilization	50-80%	20
Self-Adaptation Fitness Unit (SAFU):		**75.5-81.5**

The main issue in the table 6.6 is how to quantify the SAFU of each criteria of the adaptation process. The value column is taken from table 6.5 aggregation of UB+MBRL. The quantification of SAFU is becomes as follows:

$$75.5 = 90 \times 45\% + 60 \times 15\% + 80 \times 20\% + 50 \times 20\%$$

$$81.5 = 90 \times 45\% + 60 \times 15\% + 80 \times 20\% + 80 \times 20\%$$

From rainbow evaluation in [60], the SAFU was 70 and 76 from 90-95% resource overhead. The proposed framework registered better SAFU which are 75.5 and 81.5 from 50-80 resource overhead. Even if, when similar resource overhead is considered in the propose solution as Rainbow evaluation 90 and 95 overhead, the SAFU becomes

$$83.5 = 90 \times 45\% + 60 \times 15\% + 80 \times 20\% + 90 \times 20\%$$

$$84.5 = 90 \times 45\% + 60 \times 15\% + 80 \times 20\% + 95 \times 20\%$$

Consequently, DSRACC can shows better SAFU value on the worst resource overhead than the result of Rainbow framework on Znn.com application.

CHAPTER SEVEN

7. Conclusion, Recommendation and Future work

7.1. Conclusion

Systems need to pass through self-reconfiguration activity to modifying their runtime behaviors for tackling changes in the dynamic environment and user requirements to be called a self-adaptive system. Currently, software systems should be designed and developed with the capability of self-adaptation behavior because of the rapid change of the world as well as extreme changes of human needs on a day to day activity to fixes them. DSRACC has developed an extended framework of Rainbow that has only performed static adaptation on the basis of utility theory based. In contrast, DSRACC framework has additional features that perform machine learning strategies with a combination of utility based adaptation for dynamic reconfiguration. Thus, it enables a self-adaptive system modifying their architectural elements to meet specific quality goals. However, most existing self-adaptive frameworks have developed to feasible real-time embedded systems, robotics, and high processing application.

DSRACC has also worked for simple applications to use the running machine's resources efficiently. As familiars, the decision-making process is also an important process in self-adaptive systems for accomplishing self-reconfiguration action as a heart of adaptation planning. DSRACC combines the utility-based adaptation with model-based reinforcement learning algorithms to model the environment states, and action as well as for decision-making procedures during adaptations. On the basis of that, the combination of decision-making yields better results in performances and state-action selection strategies.

In addition to that, because of model-based reinforcement learning capability of predicting the next state of the system simultaneously with learning action helps a model checking processes by taking the output immediately before the system adapt itself. These features support the system to mitigate uncertainties and unpredictable circumstance by mapping to the affected action. Generally, DSRACC frameworks also answer the questions raised in software engineering self-adaptive systems development roadmap which is what to adapt, when to adapt, how to adapt? The frameworks can reconfigure architectural elements such as components and connectors, reconfiguration takes place when specified architectural constraints denied and perform

reconfiguration by using utility based and MBRL algorithms are answers of the above-raised questions. Finally, the proposed solution also has better SAFU values than the existing, Rainbow framework.

7.2. Recommendation

Since self-adaptive systems have a capability of reconfiguring itself according to the changing environments because they have a better performance and mitigate runtime problems. As a result, I would like to recommend software development companies to develop systems using learning strategies autonomic computing feedback loop organizations.

7.3. Future works

Hereby the proposed solution combines the utility-based and model-based reinforcement learning algorithms for adaptively action in a way of UB is calculated first and MBRL will be followed. Consequently learning time is decreased. It is better to see the difference in the result when MBRL is performed first followed by UB. This situation may important for prioritizing if a similar reward exists in different qualities based on their preferences.

Next, recently investigators have examined the effects of a single feedback loop on dynamic self-adaptation to achieve a single quality goal. But when multiple qualities goal wants to achieve in reconfiguration, it might be better using multiple feedback loop in the development of managing system. However, the entanglement between the functionality of the system and the number of feedback loop becomes complex, which leads to degradation of performance of the system.

To be assured on the result of this study, it would be better to apply the study on a big complex system, which required a higher budget and large operating execution environments. This helps researchers to identify the consistency of the results on the worst case of environments as compared to this one.

References

[1] M. U. Iftikhar and D. Weyns, "ActivFORMS: A runtime environment for architecture-based adaptation with guarantees," *Proc. - 2017 IEEE Int. Conf. Softw. Archit. Work. ICSAW 2017 Side Track Proc.*, pp. 278–281, 2017.

[2] S. Cheng, "Rainbow : Cost-Effective Software," *Innovation*, vol. 0389, no. May, p. 220, 2008.

[3] T. Vogel and H. Giese, "A language for feedback loops in self-adaptive systems: Executable runtime megamodels," *ICSE Work. Softw. Eng. Adapt. Self-Managing Syst.*, no. 3, pp. 129–138, 2012.

[4] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM*, vol. 55, no. 9, p. 69, 2012.

[5] M. Luckey and G. Engels, "High-quality specification of self-adaptive software systems," *ICSE Work. Softw. Eng. Adapt. Self-Managing Syst.*, pp. 143–152, 2013.

[6] T. Vogel and H. Giese, "Model-Driven Engineering of Self-Adaptive Software with EUREMA," vol. 8, no. 4, pp. 1–33, 2018.

[7] J. M. Franco, F. Correia, R. Barbosa, M. Zenha-Rela, B. Schmerl, and D. Garlan, "Improving self-adaptation planning through software architecture-based stochastic modeling," *J. Syst. Softw.*, vol. 115, pp. 42–60, 2016.

[8] M. Amoui, M. Derakhshanmanesh, J. Ebert, and L. Tahvildari, "Achieving dynamic adaptation via management and interpretation of runtime models," *J. Syst. Softw.*, vol. 85, no. 12, pp. 2720–2737, 2012.

[9] H. N. Ho and E. Lee, "Model-based reinforcement learning approach for planning in self-adaptive software system," pp. 1–8, 2015.

[10] R. Lemos, "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap (Draft Version of May 20, 2011)," *Springer*, no. October 2010, 2011.

[11] H. Gomaa and M. Hussein, "Software reconfiguration patterns for dynamic evolution of software architectures," *Proc. - Fourth Work. IEEE/IFIP Conf. Softw. Archit. (WICSA 2004)*, pp. 79–88, 2004.

[12] N. D'ippolito *et al.*, "Control Strategies for Self-Adaptive Software Systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 11, no. 4, pp. 1–31, 2017.

[13] S. Loukil, S. Kallel, and M. Jmaiel, "An approach based on runtime models for developing dynamically adaptive systems," *Futur. Gener. Comput. Syst.*, vol. 68, pp. 365–375, 2017.

[14] J. Swanson, M. B. Cohen, J. Firestone, B. J. Garvin, and M. B. Dwyer, "Beyond the rainbow: self-adaptive failure avoidance in configurable systems," pp. 377–388, 2014.

[15] I. Dario and P. Anaya, "systems To cite this version : Ivan Dario Paez Anaya Analysis in Self-Adaptive Pervasive Systems Intégration de," 2016.

[16] D. Yu, Q. Li, L. Wang, and Y. Lin, “An agent-based self-adaptive mechanism with reinforcement learning,” *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 3, pp. 582–585, 2015.

[17] D. Kim and S. Park, “Reinforcement Learning-Based Dynamic Adaptation Planning Method for,” *Seams*, pp. 76–85, 2009.

[18] A. Saadi, M. Oussalah, A. Henni, and D. Bennouar, “Handling the Dynamic Reconfiguration of Software Architectures using Intelligent Agents,” pp. 1–5, 2015.

[19] S. Mahdavi-Hezavehi, P. Avgeriou, and D. Weyns, “A Classification Framework of Uncertainty in Architecture-Based Self-Adaptive Systems With Multiple Quality Requirements,” *Manag. Trade-Offs Adapt. Softw. Archit.*, pp. 45–77, 2016.

[20] R. Capilla, J. Bosch, P. Trinidad, A. Ruiz-Cortés, and M. Hinchey, “An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry,” *J. Syst. Softw.*, vol. 91, no. 1, pp. 3–23, 2014.

[21] E. Albassam, J. Porter, H. Gomaa, and D. A. Menasce, “DARE: A Distributed Adaptation and Failure Recovery Framework for Software Systems,” *Proc. - 2017 IEEE Int. Conf. Auton. Comput. ICAC 2017*, pp. 203–208, 2017.

[22] K. Markhan, “Introduction to machine learning with scikit-learn,” *Kaggle’s blog*, 2015.

[23] D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste, “Rainbow: Architecture-based self-adaptation with reusable infrastructure,” *Computer (Long. Beach. Calif).*, vol. 37, no. 10, pp. 46–54, 2004.

[24] S. Hallsteinsen *et al.*, “A development framework and methodology for self-adapting applications in ubiquitous computing environments,” *J. Syst. Softw.*, vol. 85, no. 12, pp. 2840–2859, 2012.

[25] F. Kneer and E. Kamsties, “A framework for prototyping and evaluating self-adaptive systems - A research preview,” *CEUR Workshop Proc.*, vol. 1564, 2016.

[26] N. Esfahani, A. Elkhodary, and S. Malek, “A Learning-Based Framework for Engineering Software Systems,” *Ieee Trans. Softw. Eng.*, vol. 39, no. 11, pp. 1467–1493, 2013.

[27] S. Mahdavi-Hezavehi, V. H. S. Durelli, D. Weyns, and P. Avgeriou, “A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems,” *Inf. Softw. Technol.*, vol. 90, pp. 1–26, 2017.

[28] L. Castañeda, “A Reference Architecture for Software Systems,” 2012.

[29] P. Arcaini and P. Scandurra, “Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation - IEEE Xplore Document.”

[30] T. Hester, M. Quinlan, and P. Stone, “RTMBA: A real-time model-based reinforcement learning architecture for robot control,” *Proc. - IEEE Int. Conf. Robot. Autom.*, no. May, pp. 85–90, 2012.

[31] A. Metzger, A. M. Sharifloo, K. Pohl, L. Baresi, and C. Quinton, “Learning and evolution in

dynamic software product lines," pp. 158–164, 2016.

[32] M. Abufouda, "Quality-Aware Approach for Engineering Self-Adaptive Software Systems," pp. 173–181, 2014.

[33] R. Allen, R. Douence, and D. Garlan, "Specifying and analyzing dynamic software architectures," pp. 21–37, 2006.

[34] A. Bennaceur, D. Giannakopoulou, and R. Hähnle, "Machine Learning for Dynamic Software Analysis: Potentials and Limits," vol. 11026, no. 4, pp. 161–173, 2018.

[35] M. Van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," *Adapt. Learn. Optim.*, vol. 12, pp. 3–42, 2012.

[36] B. Wu, Y. Feng, and H. Zheng, "Model-based Bayesian Reinforcement Learning in Factored Markov Decision Process," *J. Comput.*, vol. 9, no. 4, pp. 845–850, 2014.

[37] C. Publishers, "csaba_RLAlgsInMDPs-lecture," pp. 1–98, 2010.

[38] F. Boyer, O. Gruber, and D. Pous, "A robust reconfiguration protocol for the dynamic update of component-based software systems," *Softw. - Pract. Exp.*, vol. 47, no. 11, pp. 1729–1753, 2017.

[39] E. Cavalcante, T. Batista, and F. Oquendo, "Supporting Dynamic Software Architectures: From Architectural Description to Implementation," *Proc. - 12th Work. IEEE/IFIP Conf. Softw. Archit. WICSA 2015*, pp. 31–40, 2015.

[40] D. Weyns and T. Ahmad, "Claims and evidence for architecture-based self-adaptation: A systematic literature review," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7957 LNCS, pp. 249–265, 2013.

[41] F. Faniyi, P. R. Lewis, R. Bahsoon, and X. Yao, "Architecting self-aware software systems," *Proc. - Work. IEEE/IFIP Conf. Softw. Archit. 2014, WICSA 2014*, pp. 91–94, 2014.

[42] P. Zoghi, M. Shtern, and M. Litoiu, "Designing search based adaptive systems: a quantitative approach," pp. 7–16, 2014.

[43] G. Tamura, N. M. Villegas, H. A. Muller, L. Duchien, and L. Seinturier, "Improving context-awareness in self-adaptation using the DYNAMICO reference model," *ICSE Work. Softw. Eng. Adapt. Self-Managing Syst.*, no. i, pp. 153–162, 2013.

[44] C. Ferreira, S. Sargento, and A. Oliveira, "An architecture for a learning-based autonomic decision system," *J. Comput. Sci.*, vol. 22, pp. 268–282, 2017.

[45] J. C. Moreno, A. Lopes, D. Garlan, and B. Schmerl, "Impact models for Architecture-based self-adaptive systems," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8997, pp. 89–107, 2015.

[46] M. Zuñiga-Prieto, J. Gonzalez-Huerta, S. Abrahao, and E. Insfran, "Framework for dynamic architecture reconfiguration of cloud services," *CEUR Workshop Proc.*, vol. 1258, pp. 46–50, 2014.

[47] R. Raheja, S. W. Cheng, D. Garlan, and B. Schmerl, “Improving architecture-based self-adaptation using preemption,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6090 LNCS, pp. 21–37, 2010.

[48] J. Panerati *et al.*, “On self-adaptive resource allocation through reinforcement learning,” *Proc. 2013 NASA/ESA Conf. Adapt. Hardw. Syst. AHS 2013*, pp. 23–30, 2013.

[49] C. Salinesi, R. Mazo, D. Diaz, V. Pelechano, and G. H. Alférez, “Dynamic adaptation of service compositions with variability models,” *J. Syst. Softw.*, vol. 91, pp. 24–47, 2013.

[50] T. Chen, R. Bahsoon, and X. Yao, *A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems*, vol. 1, no. 1. 2016.

[51] Z. Ding, Y. Zhou, and M. Zhou, “Modeling Self-Adaptive Software Systems with Learning Petri Nets,” *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 46, no. 4, pp. 483–498, 2016.

[52] B. Schmerl, G. A. Moreno, A. Mellinger, J. Camara, and D. Garlan, *Architecture-Based Self-Adaptation for Moving Target Defense*, no. 201717. 2014.

[53] D. Perez-Palacin, R. Mirandola, and J. Merseguer, “On the relationships between QoS and software adaptability at the architectural level,” *J. Syst. Softw.*, vol. 87, no. 1, pp. 1–17, 2014.

[54] K. Angelopoulos, V. E. Silva Souza, and J. Pimentel, “Requirements and architectural approaches to adaptive software systems: A comparative study,” *ICSE Work. Softw. Eng. Adapt. Self-Managing Syst.*, pp. 23–32, 2013.

[55] Y. Abuseta and K. Swesi, “Design Patterns for Self Adaptive Systems Engineering,” *Int. J. Softw. Eng. Appl.*, vol. 6, no. 4, pp. 11–28, 2015.

[56] Y. Tahara, A. Ohsuga, and S. Honiden, “Formal Verification of Dynamic Evolution Processes of UML Models Using Aspects,” *Proc. - 2017 IEEE/ACM 12th Int. Symp. Softw. Eng. Adapt. Self-Managing Syst. SEAMS 2017*, pp. 152–162, 2017.

[57] A. Elkhodary, N. Esfahani, and S. Malek, “FUSION : A Framework for Engineering Self-Tuning Self-Adaptive Software Systems,” *Int. Symp. Found. Softw. Eng.*, pp. 7–16, 2010.

[58] C. R. Sims, “Reinforcement Learning : Model-based,” pp. 1–4, 2012.

[59] E. Alpaydin, *Introduction to Machine Learning Second Edition*. 2010.

[60] S. W. Cheng, D. Garlan, and B. Schmerl, “Evaluating the effectiveness of the rainbow self-adaptive system,” *Proc. 2009 ICSE Work. Softw. Eng. Adapt. Self-Managing Syst. SEAMS 2009*, pp. 132–141, 2009.

[61] E. Rutten, N. Marchand, D. Simon, E. Rutten, N. Marchand, and D. Simon, “Computing Feedback Control as MAPE-K loop in Autonomic Computing,” 2015.

[62] B. Coronado, E. Gustafson, J. Reeder, and D. S. Lange, “Mixing Formal Methods , Machine Learning , and Human Interaction Through an Autonomics Framework,” no. Cheng 2008, pp. 175–178, 2016.

[63] K. Baclawski, E. S. Chan, D. Gawlick, A. Ghoneimy, K. C. Gross, and Z. H. Liu, “Self-

adaptive dynamic decision making processes," *2017 IEEE Conf. Cogn. Comput. Asp. Situat. Manag. CogSIMA 2017*, 2017.

[64] C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli, "Managing non-functional uncertainty via model-driven adaptivity," *Proc. - Int. Conf. Softw. Eng.*, pp. 33–42, 2013.

[65] I. Gerostathopoulos *et al.*, "Self-adaptation in software-intensive cyber–physical systems: From system goals to architecture configurations," *J. Syst. Softw.*, vol. 122, pp. 378–397, 2016.

Appendixes

Appendix 1: Java Source code

Appendix 1.1: Java User Interface code

```
class MainUIStage
 {

      StageVideo stageV;

      StageImage stageI;

      ServerControl serverControl;

   void start(Stage stage)
   {
      serverControl.startServer(num);
      stageV.start();
      stageI.start();
   }
}


class ServerControl
{
      PrimaryRegistry registry;
      MonitoringClass sysMonitoring;
      int startServer(int num)
   {
                registry.activeServer(num);
                serverNumPass(num)
                return num;
}
      int closeServers(int num)
   {
                 Registry.closeServer(num);
                 serverNumPass(num);
                 return num;
   }
      int serverNumPass(int num)
      {
                 numNumber=num;
                 sysMonitoring.numServer(numNumber);
                 return numNumber;
      }
}


class StageImage
{
      MonitoringClass sysMonitoring;
      public void start(Stage stage)
   {
                 //image displaying interface
```

```
    }
    void runtimeMetrics(long du, String mode
    {
        sysMonitoring.modeAnalysis(mode);
        sysMonitoring.getExecutionTime(du);
    }
}
class StageVideo
{
    MonitoringClass sysMonitoring;
    public void start(Stage stage)
  {
        //video playing interface
    }
    void runtimeMetrics(long du, String mode
    {
        sysMonitoring.modeAnalysis(mode);
        sysMonitoring.getExecutionTime(du);
    }
}
```

Appendix 1.2: MAPE-K source code

```
class MonitoringClass
{
    AnalysisClass analysis;
    Static int serverNumber;
    Static String deliveryMode;
    double getMemoryUsage()
    {
        double usgaeMemory;
        usageMemory  = (int)(usedMemory*100/physicalMemorySize);
        return usageMemory;
    }
    double getCPUUsage()
    {
        double cpuPercent;
        getMemoryUsage();
        cpuPercent = calcCPU(startCPUTime, start, cpuCount);
        return cpuPercent;
    }
    static int calcCPU(long cpuStartTime, long elapsedStartTime, int cpuCount)
    {
        float per = ((float)totalUsedCPUTime*100)/(float)totalAvailCPUTime);
        Return (int) per;
    }
    void getNumServer(int num)
    {
        //display active and down server
        serverNumber = num;
    }
    void getModeAnalysis(String mode)
    {
        //display what mode are available currently
        deliveryMode = mode;
```

```
    }
    void getExecutionTime(long time)
    {
            getMemoryUsage();
            getCPUUsage();
            notifyAnalysis(time);
    }
    public void notifyAnalysis(long time)
    {
             analysis.anaRes(time,cpuPercent,usageMemory,serNum,delMode);
             modelClass.updateModel(time,cpuPercent,usageMemory,serNum,delerMode);
    }
}


class AnalysisClass
{
    PlanningClass planning;
    MonitoringClass sysMonitoing;
    ServerControl openS;
    StageVideo sVideo;
    long MAX_T = 5;
    double MIN_LOAD_PER = 30;

    void anaRes(long exeTime, int cpuU, double, memoryU, int  numS, String mode)
    {
            modelClass.updateModel();
            double totalLoad = (cpuUsage + memoryUsage)/2;
            //specify architecture constraints
            if(exeTime < MAX_T && totalLoad <= MIN_LOAD_PER && numServer==1 &&
              mode==”Video”)
            {
                    //no need of reconfiguration
                    openS.serverNumPass(numServer)
                    sVideo.runtimeMetrics(exeTime,mode);
            }
            else
                    planning.startUtility(exeTime,totalLoad,numServer,mode);
    }}

class PlanningClass
{
    ModelClass modelClass;
    ExecutorClass execute;
    public void startUtility(int exeTiVal, float loadVal, int serNum,String mode)
    {
        calculateUtility(exeTiVal,loadVal,serNum,mode)
            modelClass.checkModel(utilityA,utilityD,utilityI,utilityV);
            triggerdLearningPython();
  }

    public void reconfigureAction()
    {
             execute.addserver();
             execute.downserver();
             execute.setimage();
             execute.setvideo();
```

```
    }
}

class ExecutorClass
{
    StageVideo showVideo;
    StageImage showImage;
    serverControl serverControl;

    void setImage()
    {
            showVideo.close();
            showImage.start();
    }
    void setVideo()
    {
             showImage.close();
             showVideo.start();
    }
    void setAddServer()
    {
             serverControl.startServer();
    }
    void setDownServer()
    {
             serverControl.closeServers();
    }
}
```

Appendix 2: Python source code

Appendix 2.1: Python code of State and Action Matrix

Define all available state and action from the matrix:

```
  def available_actions(state):
current_state_row = R[state,]
av_act = np.where(current_state_row >= 0)[1]
return av_act
      ADD = 0
      DOWN = 1
      VIDEO = 2
      IMAGE = 3

      ACTIONS = [ADD, DOWN, VIDEO, IMAGE]
      def collect_environmental_data(action):
         found = []
         if action in addRes:
            found.append('AD')

         if action in mode:
```

```
            found.append('IV')
        return (found)
```

Appendix 2.2: Python code for Update the Model

```
def update(current_state, action, gamma):

    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]

    Q[current_state, action] = R[current_state, action] + gamma * max_value
    print('max_value', R[current_state, action] + gamma * max_value)

    if (np.max(Q) > 0):
        return(np.sum(Q/np.max(Q)*100))
    else:
        return (0)
```

Appendix 2.3: Python code for RL

```
def update(current_state, action, gamma):

    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]

    Q[current_state, action] = R[current_state, action] + alpha * (reward + gamma *
np.max(Q[state,action]) - Q[state, action])
    print('max_reward_value', R[current_state, action] + alpha * (reward + gamma *
np.max(Q[state,action,]) - Q[state, action]))
```

Appendix 2.4: Python code for Representing Environments

```
loadVal = float(sys.argv[1])#
exeTimeVal = int(sys.argv[2]) #"VMN"
delMode = str(sys.argv[3])
initialState = str(sys.argv[4])
goalState = str(sys.argv[4])

if((delMode=="Video") and (loadVal>=1 and loadVal<=30) and (exeTimeVal<=4)):

STATE1 = "State1"
elif((delMode=="Video") and (loadVal>=1 and loadVal<=30) and (exeTimeVal>=4)):
STATE2 = "State2"
elif((delMode=="Video") and (loadVal>=31 and loadVal<=65) and (exeTimeVal<=4)):
STATE3 =   "State3"
elif((delMode=="Video") and (loadVal>=31 and loadVal<=65) and (exeTimeVal>=4)):
STATE4 =   "State4"
elif((delMode=="Video") and (loadVal>=66 and loadVal<=100) and (exeTimeVal<=4)):
STATE5 =   "State5"
elif((delMode=="Video") and (loadVal>=66 and loadVal<=100) and (exeTimeVal>=4)):
STATE6 =   "State6"
elif((delMode=="Image") and (loadVal>=1 and loadVal<=30) and (exeTimeVal<=4)):
STATE7 =   "State7"
elif((delMode=="Image") and (loadVal>=1 and loadVal<=30) and (exeTimeVal>=4)):
STATE8 =   "State8"
elif((delMode=="Image") and (loadVal>=31 and loadVal<=65) and (exeTimeVal<=4)):
STATE9 =   "State9"
elif((delMode=="Image") and (loadVal>=31 and loadVal<=65) and (exeTimeVal>=4)):
STATE10 =   "State10"
elif((delMode=="Image") and (loadVal>=66 and loadVal<=100) and (exeTimeVal<=4)):
STATE11 =   "State11"
else:
STATE12 =   'State12'
```

Appendix 3: Extracted knowledge from the Literatures

Appendix 3.1: Learning and Model development

Table 0.1 Model and learning development knowledge from literatures

ID	*Learning Algorithm/Planning strategies/techniques*	*Model/mathematical description*	*Quiescent state design*
R1	Planning algorithm - Serene greedy - Bruit-force Utility based adaptation of runtime behavior	Model description using framework and development methodology	Left to developer
R2	Dynamic decision network	Only runtime and development time framework	no
R3	NO	No mathematical description	no
R4	Online learning based algorithm		no
R5	Reinforcement learning	Mathematic description For model-based and free	no
R6	- Network of timed automata - Time computational tree logic	No, only framework design and description	no
R7	Reasoning based algorithm Statistical based machine learning	Mathematical description for candidate ranking	no
R8	programmed	clustering	no
R9	Reinforcement learning kinds which is Q-learning	Q-learning mathematical description	no
R10	- Comparison between rainbow and proposed framework - Provides strategies for unpredictable circumstances	UML model description	no
R11	- Learning via meta level - Online learning algorithm - Strategy based	no	no
R12	- Search based algorithm	Mathematical description for rank point	no
R13	- Only IDL description using diagram which shows the reconfiguration process	No description	no
R14	- Programmed fashion learning	No description	no
R15	- No learning algorithm	QA data variables Utility function	no
R16	- Prediction the impact of each strategies/actions	Description of utility, and component property using	no

	- Self-adaptive planning algorithm which is based on MDP and involves utility theory -	mathematical description	
R17	- Policy-based learning - Markov decision process with hyper state	Mathematical description with graph analysis for learning and representation	no
R18	-		
R19	- Neural network with offline learning - Dynamic decision network, behavioral adaptation - Specification methodologies by proposing model -	Enough mathematical description for learning, representation and model analysis	No state
R20	- no	no	no
R21	- Based on reinforcement learning which is Q-learning - Learning strategy - online-	Compute impact of adaptation to analysis the reward of each action	Cite previous paper rather than including in the works
R22	-		
R23	- Bayesian probability learning	Quantitative verification of runtime model and requirement model	no
R24	- No machine learning - Programmed strategies based on cost-benefit attributes -	no	no
R25	- Programmed fashion adaptation	Only aspect-oriented coding using PRISMA	no
R26	- Machine learning and game theory	Mathematical description with the framework analysis and evaluation	no
R27	- No learning capability	Has well proved and represented mathematical formulas for each adaptation impact	no
R28	- No learning	Mathematical description for utility function	no
R29	- Adaptive dynamic programming and reinforcement learning	Mathematical description for model-based learning	No
R30	- Machine learning strategies	No description	no

Appendix 3.2: Decision making and assurance ways

Table 0.2 Decision-making and assurance ways in SAS

ID	*Assurance way*	*Stochastic behavior analysis*	*Exploitation-exploration*	*Centralized/decentralized*	*Reconfiguration pattern*
R1	Assurance through testing Runtime verification	- No stochastic analysis	- Exploitation	- centralized	Left to the developer
R2	No assurance	No stochastic	No	- centralized	No description
R3	No assurance	no	no	Not mentioned	No description
R4	Simulation, testing	QA data variable Utility function	No	- centralized	no
R5	Testing using simulation	Using MDP	No trade-off analysis	- in both	no
R6	Using model-based testing	Applying statistical runtime model checking	no	-	-
R7	testing	Non description	no	both	no
R8	No assurance	Runtime models and metrics QA data variability Other model	no	centralized	
R9	Through manual testing, which is human-driven assurance	no	Only exploration of RL	- centralized distributed control	no
R10	- verification	No stochastic analysis	No exploit-explore strategies	No	No reconfiguration pattern
R11	- no	Utility function	-	decentralized	no
R12	- simulation	QA data variable	no	both	no
R13	- No any assurance way for reconfiguration	no	no	Centralized	- Left to software architect
R14	- Model checking	Non determined	no	Not mentioned	no
R15	- simulation	QA data variability	no	Not mentioned	no
R16	- Testing using metrics	- Analysis based on DTMP	no	No mentioned	
R17	- No assurance way	With MDP	Trade-off solved	No description	No description
R18	-	no	no	centralized	no
R19	- No assurance way	No stochastic behavior analysis	No	No description	No description

R20	- testing	other	no	no	no
R21	- Robocode testing	It describes strategy for exploit-explore trade-off	Both trade-off solved by e-greedy	No description	No pattern for reconfiguration
R22	-				
R23	- verification	no	Only exploitation	No controlled data	No pattern
R24	- no assurance way	Stochastic model rather than behavior with utility theory	no	External controller	No pattern
R25	- no	no	no	decentralized	no
R26	- assurance using model checking tool	Probabilistic model checking tools for stochastic behavior	Use exploitation-exploration without trade-off managing	simulation	
R27	- no	Uses DTMD	no	no	no
R28	- no	no	no	no	no
R29	- Simulation testing	Uses MDP for stochastic analysis	Uses only exploitation	Centralize model managing	No reconfiguration pattern
R30	- Assurance through testing	no	Exploration only	centralized	No pattern

Appendix 3.3 Selected Literature for SLR

Table 0.3 Selected literatures for SLR

Paper#	*Author(s)*	*Year*	*Title*
R24	Hallsteinsen, Svein, Kurt Geihs, Nearchos Paspallis, Frank Eliassen, Geir Horn, Jorge Lorenzo, Alessandro Mamelli, and George Angelos Papadopoulos	2012	A development framework and methodology for self-adapting applications in ubiquitous computing environments.
R25	Kneer, Fabian, and Erik Kamsties	2016	A Framework for Prototyping and Evaluating Self-adaptive Systems-A Research Preview
R3	Vogel, Thomas, and Holger Giese	2012	A language for feedback loops in self-adaptive systems: Executable runtime megamodels
R26	Esfahani, Naeem, Ahmed Elkhodary, and Sam Malek	2013	A learning-based framework for engineering feature-oriented self-adaptive software systems
R30	Hester, Todd, Michael Quinlan, and Peter Stone	2012	RTMBA: A real-time model-based reinforcement learning architecture for robot control
R1	Iftikhar, Usman, and Danny Weyns	2017	ActivForms: A Runtime Environment for Architecture-Based Adaptation with Guarantees
R32	Abufouda, Mohammed	2017	Quality-aware Approach for Engineering Self-adaptive Software Systems
R8	Amoui, Mehdi, Mahdi Derakhshanmanesh, Jürgen Ebert, and Ladan Tahvildari	2012	Achieving dynamic adaptation via management and interpretation of runtime models
R44	Ferreira, Carlos, Susana Sargento, and Arnaldo Oliveira	2017	An architecture for a learning-based autonomic decision system
R13	Loukil, Sihem, Slim Kallel, and Mohamed Jmaiel	2016	An approach based on runtime models for developing dynamically adaptive systems
R41	Faniyi, Funmilade, Peter R. Lewis, Rami Bahsoon, and Xin Yao	2014	Architecting self-aware software systems
R42	Zoghi, Parisa, Mark Shtern, and Marin Litoiu	2014	Designing search based adaptive systems: a quantitative approach
R46	Zuñiga-Prieto, Miguel, Javier Gonzalez-Huerta, Silvia Mara Abrahão, and Emilio Insfran	2014	Framework for Dynamic Architecture Reconfiguration of Cloud Services.
R5	Luckey, Markus, and Gregor Engels	2013	High-quality specification of self-adaptive software systems
R43	Tamura, Gabriel, Norha M. Villegas, Hausi A. Müller, Laurence Duchien, and Lionel Seinturier	2013	Improving context-awareness in self-adaptation using the DYNAMICO reference model
R7	Franco, João M., Francisco Correia,	2016	Improving self-adaptation planning

	Raul Barbosa, Mário Zenha-Rela, Bradley Schmerl, and David Garlan		through software architecture-based stochastic modeling
R9	Ho, Han Nguyen, and Eunseok Lee	2015	Model-based reinforcement learning approach for planning in self-adaptive software system
R6	Vogel, Thomas, and Holger Giese.	2014	Model-driven engineering of self-adaptive software with eurema
R51	Ding, Zuohua, Yuan Zhou, and MengChu Zhou.	2014	Modeling self-adaptive software systems with learning Petri nets
R53	Perez-Palacin, Diego, Raffaela Mirandola, and José Merseguer.	2013	On the relationships between QoS and software adaptability at the architectural level
R17	Kim, Dongsun, and Sooyong Park.	2009	Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software
R54	Angelopoulos, Konstantinos, Vítor E. Silva Souza, and João Pimentel.	2013	Requirements and architectural approaches to adaptive software systems: A comparative study
R4	Calinescu, Radu, Carlo Ghezzi, Marta Kwiatkowska, and Raffaela Mirandola.	2012	Self-adaptive software needs quantitative verification at runtime.
R2	Garlan, David, Bradley R. Schmerl, and Shang-Wen Cheng.	2009	Software Architecture-Based Self-Adaptation
R18	Costa-Soria, Cristóbal, Jennifer Pérez, and Jose Ángel Carsí.	2009	Handling the dynamic reconfiguration of software architectures using aspects
R52	Schmerl, Bradley, Javier Camara, Gabriel Moreno, David Garlan, and Andrew O. Mellinger.	2010	Architecture-Based Self-Adaptation for Moving Target Defense (CMU-ISR-14-109)
R45	Moreno, Javier Cámara, Antónia Lopes, David Garlan, and Bradley Schmerl.	2011	Impact models for architecture-based self-adaptive systems
R47	Raheja, Rahul, Shang-Wen Cheng, David Garlan, and Bradley Schmerl.	2013	Improving architecture-based self-adaptation using preemption
R48	Panerati, Jacopo, Filippo Sironi, Matteo Carminati, Martina Maggio, Giovanni Beltrame, Piotr J. Gmytrasiewicz, Donatella Sciuto, and Marco D. Santambrogio.	2014	On self-adaptive resource allocation through reinforcement learning
R67	Martina Maggio, Tarek Abdelzaher, Lukas Esterle, Holger Giese, Jeffrey O. Kephart, Ole J. Mengshoel, Alessandro V. Papadopoulos, Anders Robertsson and Katinka Wolter	2017	Self-adaptation for Individual Self-aware Computing Systems