SELF-AWARE ADAPTABLE SOFTWARE ARCHITECTURAL PATTERN BASED ON ASSOCIATION RULE MINING

Rajith Balakrishnan
Research Scholar
Anna University, Chennai, India
ranjithbke@gmail.com

Dr. N. Sankarram
Professor, Computer Science and Engg,
Sriram Engineering College
Chennai, India
n_sankarram@yahoo.com

***Abstract*-- Software architecture is one of the evergreen research areas, as the technical advancement relies on the software architecture. Due to the technical advancements, most of the users possess different terminals for accessing Internet. Irrespective of the diversity in operating system and user terminal, the web service must be interoperable and compatible. Based on the kind of user terminal, the web service has to be provided without any time complexity. The performance of the architecture is claimed better, only when the appropriate web service is provided to the user without any time delay. In order to map appropriate web services for varying user terminal, this article presents a self-aware adaptable software architectural pattern that is based on association rule mining. The proposed approach does not require any prior knowledge about the system and can work on the go. The performance of this approach is analysed by varying the count of user requests and time with respect to accuracy, precision, recall, F-measure and throughput. The proposed approach proves nominal results with better dynamic mapping capability.**

***Keywords*-- Software architecture, adaptable software, web application.**

1. INTRODUCTION

Software architecture is the blueprint of any software to be developed and hence, the quality of the software can be determined with the software architecture. The complete structure and the functionality of the software are portrayed in software architecture. The key objective of this representation is to detect the significant components involved in the software system and their interrelationship as well. As stated earlier, software architecture outlines all the components involved in the system along with their functional details in an abstract form. Hence, it is very important to have clear-cut and sharp architecture, as the positive progress of software development relies on a better software architectural design. There are numerous formal definitions and representational tools for software architecture in the literature. However, each and every technique has its own merits and demerits.

Initially, the requirements of a client are drafted formally by means of the process of requirements engineering. When this process is completed successfully, then the requirements of the software are drafted in a software requirements specification document. As soon as the requirements are finalized and upon the positive approval of clients, the design of the software is decided by means of software architecture. The software architectural representation sketches the different software components and their interconnections with each other. Hence, a good software architectural design has to impart better knowledge to the user. All these points are easy to follow, when the system is static.

Due to the advent of technology, most of the software systems are dynamic today. The major reason is the usage of different kinds of smart terminals such as mobile phones, Personal Computer (PC), laptops, PDAs and so on. As the applicability of software differs in terms of underlying platform, it is very difficult to manage. In this case, presenting software architecture is a highly challenging task. For instance, the software must be adaptable to different environments and must be capable of discovering and provisioning the available services. In all these cases, the Quality of Service (QoS) must not be compromised.

Taking these issues into account, this research work aims to present a robust self-adaptive, and a dynamic adaptable software architectural pattern. The goal of this work is to make the system function without any hassles, while the behavioural or structural pattern of the environment changes. This is made possible by designing an architectural solution that considers several different scenarios. The term ‘adaptive or adaptable’ is very generic, as it conceives different meanings. This work uses the term adaptive or adaptable to emphasize that the proposed architectural pattern can change its nature with respect to the varying terminals. The highlighting points of this work are as follows.

- This work considers different user terminals, which can connect to internet and different operating systems such as iOS, blackberry, android, windows and so on.
- This work differentiates between the type of user terminals and operating systems by means of building association rules.
- As the decision is made by association rules, the decision accuracy is perfect and the Quality of Service (QoS) is better.
- The performance of the proposed approach is evaluated by means of throughput by varying the number of transactions and the count of transactions processed in a stipulated period of time.

The remainder of this paper is organized as follows. Section 2 presents the review of literature with respect to adaptable software systems. The proposed software architectural pattern is presented in section 3. Section 4 analyses the performance of the proposed approach and the results are discussed. The concluding remarks are presented in section 5.

2. Review of Literature

Frequency representation are mostly preferred for this section reviews and discusses about the state-of-the-art adaptable software systems in the literature.

In [1], a self-adaptive software system is designed by means of Petri Nets. This work employs adaptive Petri net for developing the adaptive software system. The computation of this work is carried out on a local component, though the complete system is adapted. However, the main issue of this work is that it consumes more time to process the user requests. A fuzzy based self-adaptive software system is modelled by means of extended Unified Modelling Language (UML) in [2]. The proposed extended UML is based on fuzzy concepts. This fuzzy based UML can be incorporated in the UML development environments. However, this work suffers from technical complexity.

In [3], a technique based on UML is proposed for designing self-adaptive software system and the proposed language is termed as Adapt Case Modeling Language (ACML). This work enables the self-adaptiveness for software by means of adaptation rules and deadlock freedom capabilities. A comparative study is presented by considering the requirements and architectural approaches of adaptive software systems in [4]. The architectural and requirement models are carried out with respect to adaptation and guidelines.

A learning based framework for designing feature oriented self-adaptive software system is presented in [5]. The adaptation knowledge is represented by feature based approach and the adaptation decision is tested over online learning based approach. The knowledge is based on feature model and the features are structured. The experimental results of this work determine the ability of the system in terms of adaptation and efficiency. In [6], the effectiveness of controllers in self-adaptive software systems is examined. Controllers are software components that make final decision by means of information forwarded by probes.

An aspect oriented framework for developing self-adaptive software system namely JACAC is proposed in [7]. The aspect oriented approach aims to segregate between the business logic and crosscutting concerns. This work clubs aspect and components for proposing the JACAC. A supporting framework for self-adaptive software system is proposed in [8]. This framework supports the adaptable software systems that can cope with environmental and user requirement changes. This framework deals with the controllable architecture, an application independent language and implementation part.

A service based technique is proposed for self-adaptive software systems in [9]. This work relies on autonomous elements that can be utilized for specific kind of service or user requirement. This work claims that it requires less memory and computational resources. An architectural model that supports adaptive software systems, which is meant for software systems is proposed in [10]. This work utilizes architecture description language for representing the software components and is reconfigured with respect to the requirements of the user. The work presented in [11] reviews the adaptive software in terms of techniques, tools and applications. This work presents the details about the techniques that change the structure and function of the software, so as to cope up with the user requirements.

A rigorous architectural reasoning framework is presented in [12] for designing self-adaptive software systems. An analytical framework that can improve the extended architectural reasoning framework is presented. The analytical framework is composed of artefacts and activities. The basic model, properties and attributes are included in the templates of artefacts. The activities transform the requirement instances to architecture models and the system is illustrated with the client-server system.

In [13], a testing scheme is proposed for self-adaptive software systems with architectural runtime model. The proposed testing scheme enables the developers to check the feedback loops in an earlier state by using the architectural runtime model. These models are utilized at runtime and the test inputs are described better. The resilience level of self-adaptive software is evaluated in [14]. The resilience level of

the software is checked by the probabilistic model checking and the properties of satisfaction are measured.

In [15], distributed self adaptive software systems are executed for adaptations in co-ordinated manner. The adaptations are checked on multiple nodes of the distributed system and the consistency is tested. This work states that it is free from deadlock scenarios. The self-adaptive software systems are modelled by fuzzy rules and petrinets in [16]. The Intelligent Petri net (I-PN) is utilized for modelling the software system. This work can model the run-time environment and the behaviour of the software system. This model can present the self-adaptive software with the ability to make decisions at runtime by means of fuzzy inference reasoning. A workflow based self-adaptive system is proposed for monitoring and predicting the software is proposed in [17]. In this model, the users can specify their requirements in the model and the QoS attributes are computed dynamically. The operating condition and the context of the software system are monitored.

Motivated by the above presented works, this article aims to present self-aware and adaptable software architectural pattern that relies on several components. The goal of this work is to ensure robustness and better QoS. The proposed approach is elaborated in the following section along with the overview of the proposed work.

3. PROPOSED SELF-AWARE ADAPTABLE SOFTWARE ARCHITECTURE

The technical and functional details of the Self-aware Adaptable Software Architecture are presented in this section. Initially, the overall flow of the system is presented.

3.1 Overall Flow of the System

The central theme of this approach is to ensure better web services to different range of user terminals, irrespective of the operating system. Nowadays, numerous counts of user terminals are in market and over ninety percent of the mobile terminals are connected to the network. Hence, the user request is submitted from varying mobile terminals to the server. Though the content of web application is the same, the server has to respond in different ways. Though this problem has many solutions in the literature, the software architecture based solutions for this issue are limited. Hence, this work proposes a self-aware adaptable software architecture system based on association rules. In order to achieve this goal, the architectural pattern involves three tiers such as user terminal, knowledge adoption and service response tier. The layered architecture of the proposed approach is depicted in figure 1.

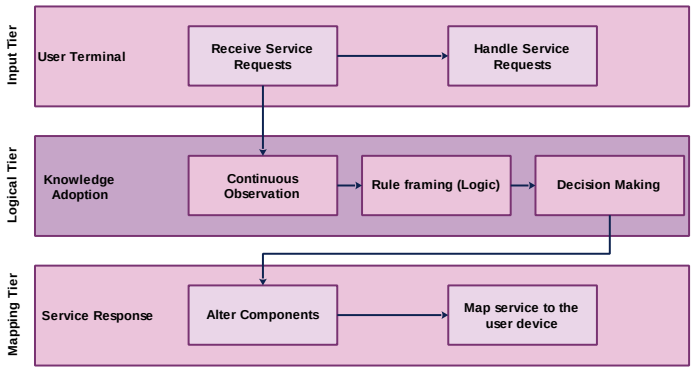

Fig.1. Layered architecture of the proposed architectural pattern

The user terminal tier can enclose different mobile terminals such as smart phones, desktops, laptops, PDAs and all the devices that can access the services of internet. With these terminals, the users submit the request to the server. Before forwarding the service request to the server, there comes an intermediate layer, which decides the look and feel of the web application in a dynamic fashion. In this case, the content and the services are the same, yet the visual look and feel of the web application differs for every terminal. Additionally, different user terminals utilize different operating systems and the web services have to adapt these changes dynamically. Finally, the service response tier provides service to the user by considering the user terminal.

The pre-defined web standards provide interoperability and compatibility to the web services, such that it adapts to the environment in a dynamic fashion. This work intends to address the same issue by means of architectural approach. Though there are several solutions for the same issue, this work attempts to incorporate association rules. Some of the important features of the proposed work are this work does not require any special training to distinguish between the user terminal and operating environment, the throughput rate is better even when the count of transactions are varied.

3.2 Proposed Approach

As discussed earlier, this work employs three different tiers such as user terminal, knowledge adoption and service response tier. All the tiers have its own purpose and work with better coordination to attain nominal QoS. The reason for the incorporation of tiered architecture is that the actions to be performed by the tiers are segregated perfectly and is easy to manage. Additionally, the tiered architecture makes it easy to modify or remove any component in the system. The following subsections present the functionality of the tiers in detail.

3.2.1 User terminal tier

The user terminal tier handles the user service requests from different user devices. Usually, the user terminals intend to submit web service request to the server. In this case, the user terminal can be any device that can connect with the internet. The web service request is submitted to the application by means of web browser. The web application has to handle service requests from different web browsers of different user terminals. Based on the requirements of the user, the service request is linked to the service response. From this simple scenario, different ways of variations such as different browsers, user terminals, operating systems are observed. Irrespective of these variations, the user must be responded with the service demanded without any time delay or performance degradation. The intermediary tier plays a vita role in determining the right kind of service to the user and is presented below.

3.2.2 Knowledge adoption tier

This tier contains the significant components, which are meant for mapping the perfect service to the user. In order to render perfect service, a component that can recognize the user's requirement and make decision on the service provision is needed. As the decision has to be made in dynamic fashion, the role of knowledge adoption layer is more. Hence, this tier has got more functionality and is achieved by three important modules such as observation, rule formation and decision module. In fine, this module brings in adaptation to the ordinary software system. These modules are the building blocks of the tier and these modules are explained below.

3.2.2.1 Observation module

This module listens to the activities that happen on the network and captures the processing traces of the system continuously. This module intends to build item set by tracking the activities in a continuous fashion. The item set is created with respect to a time period. For every time period, there occurs a set of events and are saved as item set.

Let $TP = \{tp_1, tp_2, tp_3, \dots tp_n\}$ and $tp_x = \{evt_1, evt_2, evt_3, \dots, evt_n\}$, where TP is the complete time period and $tp_1, tp_2, tp_3, \dots tp_n$ are the time slots, in which different events take place. The goal of this module is to listen to the occurrence of events taking place in every time period and create item sets. Additionally, this module performs pre-processing activity that identifies the type of user terminal by means of Wireless Universal Resource File (WURFL). The WURFL processes the header information of Hyper Text Transfer Protocol (HTTP) request and extracts the characteristic features of the user terminal. The term 'characteristic feature' denotes the model of the device, web browser, operating system, screen size. Taking these characteristic features into account, the web services can be provided in line with the user's requirement.

As soon as the characteristic features of the user terminal that involved in submitting service request are obtained, the user terminals are clustered by employing simple distance metric. The distance metric being utilized to perform clustering is the Euclidean distance measure and is computed by

$$E_d = \sum_{i=1}^{n} \sqrt{x_i^2 - y_i^2} \quad \text{---------- (1)}$$

In the above equation, x_i and y_i are the events and n is the total number of events that occurred in a specific time period. The cluster members are closely related to each other, preferably the terminal with same device configuration. From these clusters, the frequently participating user terminals in a particular period of time can be figured out easily. Similarly, the web services are clustered by following the same way. The web service clustering process encloses the web services for different terminals. For instance, the web services meant for android based mobile devices are grouped together. In addition to this, the size of the screen is also taken into account for attaining web service clusters. In the next module, the association rules are constructed on the go, such that the appropriate web service can be mapped to the user.

3.2.2.2 Rule formation (logical module)

This is the most crucial step as the rules are formed and it determines the frequent patterns. The frequent patterns are created by means of two important key ingredients, which are support and confidence. The overall functionality of this module is depicted in figure 2.

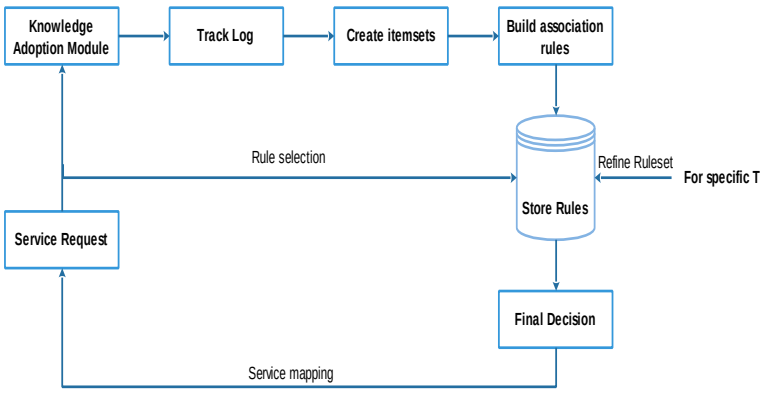

Fig.2. Functionality of knowledge adoption module

As the rule formation of this work is based on apriori algorithm, support and confidence are the main attributes. The support and confidence are computed between the event clusters and web service clusters as follows. The support of $e_a \Rightarrow e_b$ is computed by dividing the count of events of same

type that occurred both in tp_1 and tp_2 with the total number of events in the cluster.

$$SPT(e_a \Longrightarrow e_b) = \frac{|e_a \cup e_b \subset (ec,wsc)|}{|(ec,wsc)|} \quad (2)$$

Similarly, the confidence of the association rule is computed by the following equation.

$$Conf(e_a \Longrightarrow e_b) = \frac{SPT(e_a \Rightarrow e_b)}{SPT(e_a)} \quad (3)$$

The frequently occurring pattern is identified by means of considering the support and confidence. The support and confidence level of an association rule must be greater than the minimum support and confidence threshold. In this work, the minimum support threshold is fixed as 2 and the minimum confidence level is set as fifty percent. By this way, the frequent item set is detected and the rules are added to the decision database.

As the web accessing nature of users is dynamic, the computed association rules and the frequent patterns are not static and they vary by time. Hence, the rules are refined then and there, so as to ensure better service accessibility in less period of time. The process of rule refinement may involve rule inclusion, deletion and modification. The top-ranking frequent web service requests are given room in cache memory, such that the service is provided in streak. This in turn increases the performance and the QoS as well.

3.2.2.3 Decision making module

The decision making module is responsible for declaring the nature of user terminal and the web service to be provided by applying the idea of the logical module. The outcome of this module is considered as final and the web service is provided to the user accordingly. The decision of this module is passed on to the final tier, which is service response tier. The details of this tier are presented as follows.

3.2.3 Service Response Tier

This tier contains servers within which the web services reside. Based on the decision of the decision making module, the appropriate service is mapped to the user. The servers contain several functional components, which are to be altered before provisioning. On the other hand, the user terminal incudes several web page components that are enabled by different web browsers such as Google Chrome, Mozilla Firefox, Opera, Internet Explorer and so on. The service response tier is controlled by Internet Information Service (IIS), Tomcat and so on. By taking the user request and the nature of service request into account, the web service is mapped to the user terminal. This kind of architectural pattern is found to be reliable and robust, which is found by performing experiments in different scenarios. The following section analyses the performance of the proposed architectural pattern for web service access.

4. Results and Discussion

The effectiveness of the proposed approach is evaluated on a stand-alone computer with 8 GB RAM and Intel i7 processor. The proposed approach is simulated in MATLAB environment. The performance of the proposed approach is tested by varying the number of user requests and time in terms of standard performance metrics such as throughput, precision, recall, F-measure are tested are utilised to justify the efficiency of the proposed approach. The formula for computing the performance measures are presented as follows.

The throughput is the rate of service response by the web server and it measures the amount of data being transmitted with respect to time and is denoted by the following equation. Greater throughput rates indicate the better performance of the proposed approach.

$$\mathbb{T} = \frac{Data\ transmitted\ (mb)}{Time\ (Sec)} \quad (4)$$

Precision and recall are important performance measures that determine the working ability of the proposed approach. Precision rate is the fraction of total number of correctly mapped services to the total count of services being mapped. In this case, a perfect service mapping scheme attains greater precision rates, as it encounters maximal perfect service mappings. On the other hand, the recall rate is measured by computing the ratio of total number of correctly mapped services to the total count of perfect services with respect to the service request in the database. The precision (P) and recall ($\mathbb{R}$) rates are measured by the following formulas.

$$P = \frac{Total\ number\ of\ correctly\ mapped\ services}{Total\ count\ of\ mapped\ services} \quad (5)$$

$$\mathbb{R} = \frac{Total\ number\ of\ correctly\ mapped\ services}{Total\ matching\ services} \quad (6)$$

The F-measure is computed on the basis of the P and $\mathbb{R}$ values. In this case, greater F-measure means that the web services are correctly mapped to the user requests and is computed by

$$\mathrm{F} = \frac{2P\mathbb{R}}{P+\mathbb{R}} \quad (7)$$

The accuracy rate is computing the ratio of the summation true positive (TP) and true negative (TN) rates to the summation of true positive, true negative, false positive (FP) and false negative (FN) rates. The following equation presents the formula for computing the accuracy rate.

$$A = \frac{TP+TN}{TP+TN+FP+FN} \quad (8)$$

The experimental results of the proposed work are presented as follows. Initially, the performance of the proposed approach is tested by varying the count of processed user requests. The processed user requests are varied from 1 to 100.

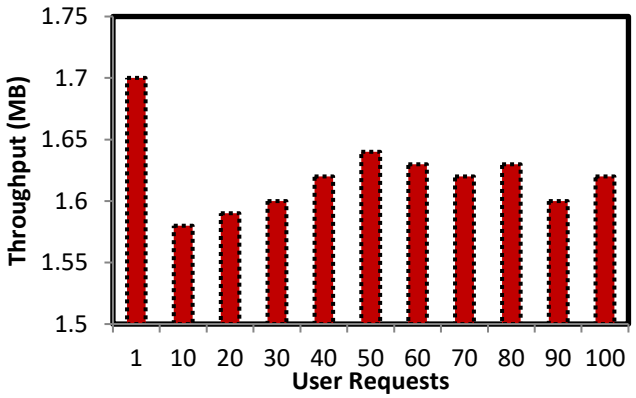

Fig.3 Throughput analysis by varying the user requests

On observation, it is noticed that the throughput of the proposed approach is satisfactory. Greater throughput rates enhance the performance of the system. This analysis is carried out by varying the count of user requests to be processed. The experimental results show that the proposed approach shows maximum throughput, which is 1.71 MB when dealing with one user request. The least throughput being shown by this approach is 1.58 MB that happens when processing ten user requests. As the number of user requests progresses, the proposed approach shows stable results that is around 1.6 MB. The following section presents the average precision, recall and F-measure of the proposed approach

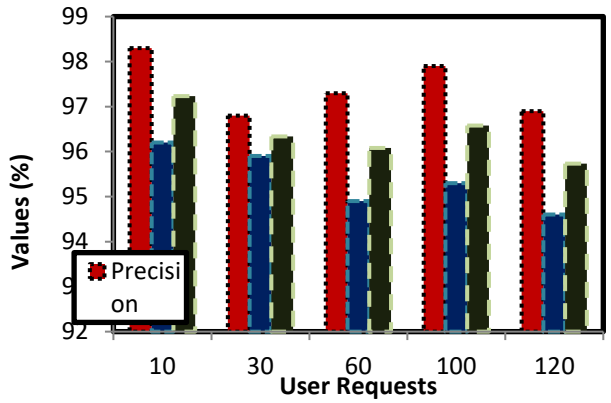

Fig.4. Comparative analysis w.r.t precision, recall and F-measure

The proposed approach is tested for its precision, recall and F-measure rates by varying the user requests from 10 to 100. The proposed approach shows consistent results, irrespective of the count of user requests. The highest precision rate being shown by the proposed approach is 98.3 percent and it happened while the system handles ten user requests. The least precision rate of the proposed approach is 96.8 percent and it occurred while handling thirty user requests. The average precision rate of the proposed approach is 97.4 percent.

When it comes to recall rates, 96.2 percent is the greatest recall rate being proven by the proposed approach. The highest recall rate is achieved while dealing with ten user requests. The least recall rate being recorded during this simulation is 94.6 percent and it happened when the system processed 120 user requests. The average recall rate of the proposed approach is 95.3 percent. The metric F-measure depends on the precision and recall measures. As this work proves better precision and recall rates, it is obvious that the F-measure of this work is greater. The following table tabulates the results attained by the proposed approach.

User Requests	Precision (%)	Recall (%)	F-Measure (%)	Accuracy (%)
10	**98.3**	**96.2**	**97.23**	**95.4**
30	96.8	95.9	96.34	96.2
60	97.3	94.9	96.08	94.4
100	97.9	95.3	96.58	93.2
120	96.9	94.6	95.73	94.8
Average	97.4	95.3	96.39	94.82

Table 1. Experimental Results by varying user requests

F-measure is directly proportional to the combination of precision and recall rates. The greatest F-measure being recorded is 97.23 percent, which is shown during the submission of ten user requests. As stated earlier, the greatest F-measure is attained while processing ten user requests, as the greatest precision and recall rates are also attained during the execution of ten user requests. 95.73 percent is the smallest F-measure rate that is observed when processing 120 user requests. In addition to this, the count of service requests being responded in a specific period of time is calculated and the experimental results are presented in figure 5.

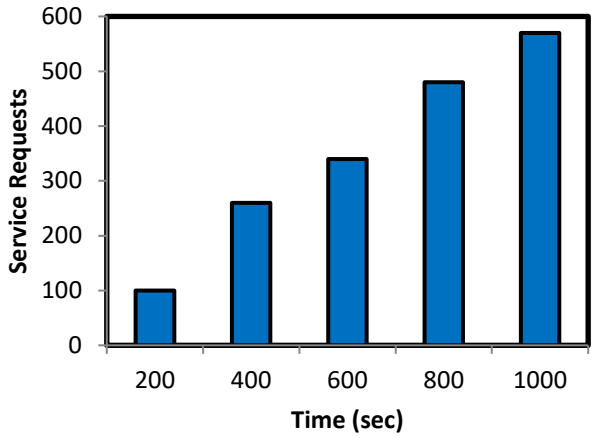

Fig.5 Service request handling analysis

Hence, the efficiency of the proposed approach is proven with greater throughput, precision, recall, F-measure and accuracy rates. The results are obtained by varying the user requests ranging from 1 to 120. The service request handling ability of the proposed approach is proven to be better, as it shows rapid progress. As time progresses, the number of service requests being handled increases. This increases the robustness and reliability of the system. The main reason for better service mapping is that the proposed approach relies on association rules, which are constructed by utilising tiered architecture. As the rules are framed dynamically, the perfect service is mapped by performing alteration. Thus, the proposed tiered architectural pattern works better, with respect to the tested scenario. Additionally, it is easy for performing any dynamic operation such as alter or delete over the components involved in the tier.

5. CONCLUSION

This article proposes a self-aware, dynamic, adaptable software architectural pattern based on association rule mining. The proposed software architectural pattern relies on three significant tiers such as user terminal tier, knowledge adoption tier and service response tier. The user terminal tier is responsible for accepting different user requests from wide range of user terminals. The knowledge adoption tier is the heart of the proposed approach that is decomposed into three modules such as observation, rule formation and decision making module. The observation module performs the preliminary tasks for framing association rules. The rules are then formed by considering the support and confidence values. Based on the user request, the decision making module commands the service response tier to map the appropriate service to the user. The dynamic service provision involves component deletion or modification. Finally, the performance of the proposed approach is evaluated by varying the user requests and the efficacy of the architectural pattern is tested in terms of throughput, accuracy, precision, recall and F-measure rates. On analysis, the proposed approach shows reasonable results and is robust. In future, this work is planned to be extended by considering dynamic user requirements, which changes the functional ability of the components.

REFERENCES

[1] Zuohua Ding; Yuan Zhou; Mengchu Zhou, "Modeling Self-Adaptive Software Systems with Learning Petri Nets", IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol.46, No.4, pp.483-498, 2016.

[2] Deshuai Han; Qiliang Yang; Jianchun Xing, "Extending UML for the modeling of fuzzy self-adaptive software systems", The 26th Chinese Control and Decision Conference, 31 May-2 June, Changsha, China, 2014.

[3] Markus Luckey ; Gregor Engels, "High-quality specification of self-adaptive software systems", ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, 20-21 May, San Francisco, USA, 2013.

[4] Konstantinos Angelopoulos ; Vítor E. Silva Souza ; João Pimentel, "Requirements and architectural approaches to adaptive software systems: A comparative study", ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, 20-21 May, San Francisco, USA, 2013.

[5] Naeem Esfahani ; Ahmed Elkhodary ; Sam Malek, "A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems", IEEE Transactions on Software Engineering, Vol.39, No.11, pp.1467-1493, 2013.

[6] Javier Cámara ; Rogério de Lemos ; Nuno Laranjeiro ; Rafael Ventura; Marco Vieira, "Robustness Evaluation of Controllers in Self-Adaptive Software Systems", Sixth Latin-American Symposium on Dependable Computing, 1-5 April, Rio de Janeiro, Brazil, 2013.

[7] Selim Kebir, "JACAC: An aspect oriented framework for the development of self-adaptive software systems", International Conference on Sciences of Electronics, Technologies of Information and Telecommunications, 21-24 Mar, Sousse, Tunisia, 2012.

[8] Liangdong Wang; Yang Gao; Chun Cao; Li Wang, "Towards a General Supporting Framework for Self-Adaptive Software Systems", IEEE Annual Computer Software and Applications Conference Workshops, 16-20 July, Izmer, Turkey, 2012.

[9] Gholamreza Safi; Seyed-Hassan Mirian-Hosseinabadi, "A Service Based Approach to Self-Adaptive Software Systems Based on Constructing a Group of Autonomic Elements", IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems, 22-26 March, Oxford, England, 2010.

[10] Hyun-Chong Kim; Ho-Jin Choi; In-Young Ko, "An architectural model to support adaptive software systems for sensor networks", 11th Asia Pacific Software Engineering Conference, 30 Nov-3 Dec, Busan, South Korea, 2004.

[11] J. Gray; R. Klefstad; M. Mernik, "Adaptive and evolvable software systems: techniques, tools, and applications", 37th Annual Hawaii International Conference on System Sciences, 5-8 Jan, Big Island, USA, 2004.

[12] Nadeem Abbas; Jesper Andersson; Muhammad Usman Iftikhar ; Danny Weyns, "Rigorous Architectural Reasoning for Self-Adaptive Software Systems", Qualitative Reasoning about Software Architectures, 5-8 April, Venice, Italy, 2016.

[13] Joachim Hänsel ; Thomas Vogel ; Holger Giese, "A Testing Scheme for Self-Adaptive Software Systems with Architectural Runtime Models", IEEE International Conference on elf-Adaptive and Self-Organizing Systems Workshops, 21-25 Sep, Cambridge, USA, 2015.

[14] Javier Cámara ; Rogério de Lemos ; Nuno Laranjeiro ; Rafael Ventura; Marco Vieira, "Robustness-Driven Resilience Evaluation of Self-Adaptive Software Systems", IEEE Transactions on Dependable and Secure Computing, Vol.14, No.1, pp.50-64, 2017.

[15] Martin Weißbach; Philipp Chrszon ; Thomas Springer ; Alexander Schill, "Decentrally Coordinated Execution of Adaptations in Distributed Self-Adaptive Software Systems", 11th International Conference on Self-Adaptive and Self-Organizing Systems, 18-22 Sep, Tucson, USA, 2017.

[16] Zuohua Ding ; Yuan Zhou ; Mengchu Zhou, "Modeling Self-Adaptive Software Systems By Fuzzy Rules and Petri Nets", IEEE Transactions on Fuzzy Systems, Vol.PP, No. 9, pp.1-1, 2017.

[17] Xiaowei Zhang; Bin Li ; Junwu Zhu, "A Monitoring and Prediction Model of Workflow Based Self-Adaptive Software System", Second International Conference on Advanced Cloud and Big Data, 20-22 Nov, Huangchen, China, 2014.