International Journal on Future Revolution in Computer Science & Communication Engineering
Volume: 4 Issue: 4

ISSN: 2454-4248
641 – 649

Leveraging Self-Adaptive Dynamic Software Architecture

1 Mr. Sridhar Gummalla,
Associate Professor, CSE,Shadan college of engineering & technology, Hyderabad, Telengana, India.
sridhargummalla1975@gmail.com}

2 Dr. G. Venkateswara Rao,
Associate Professor, Gitam Institute Of Technology, Gitam University, Visakhapatnam, A.P, India
vrgurrala@yahoo.com}

Abstract:- Software systems are growing complex due to the technological innovations and integration of businesses. There is ever increasing need for changes in the software systems. However, incorporating changes is time consuming and costly. Self-adaptation is therefore is the desirable feature of any software that can have ability to adapt to changes without the need for manual reengineering and software update. To state it differently robust, self adaptive dynamic software architecture is the need of the hour. Unfortunately, the existing solutions available for self-adaptation need human intervention and have limitations. The architecture like Rainbow achieved self-adaptation. However, it needs to be improves in terms of quality of service analysis and mining knowledge and reusing it for making well informed decisions in choosing adaptation strategies. In this paper we proposed and implemented Enhanced Self-Adaptive Dynamic Software Architecture (ESADSA) which provides automatic self-adaptation based on the runtime requirements of the system. It decouples self-adaptation from target system with loosely coupled approach while preserves cohesion of the target system. We built a prototype application that runs in distributed environment for proof of concept. The empirical results reveal significance leap forward in improving dynamic self-adaptive software architecture.

Index terms – *Self-adaptation, dynamic software architecture, reusability, maintainability*

*****

I. INTRODUCTION

Software systems drive the business in this age of digital world. The modern software systems should be equipped with highly desired features in a distributed environment. Therefore software systems must become more versatile, flexible, resilient, dependable, service-oriented, mashable, inter-operable, continuously available, robust, decentralized, energy-efficient, recoverable, customizable, configurable, self-healing, configurable and self-optimizing by adapting to changing operational contexts and environments. Traditional software is implemented under static decisions in analysis and design time based on assumptions about the requirements and runtime environment. Therefore any unanticipated changes to the requirements or runtime environment will lead to a manual maintenance process, which is unacceptable in critical systems. The existing self-adaptive software architectures are utility based and making them quality-aware is a challenging problem to be addressed. Self-adaptive software modifies its own behavior at runtime in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation. Application developers must answer several questions when developing a self-adaptive software system. Under what conditions does the system undergo adaptation? Should the system be open-adaptive or closed-adaptive? What type of autonomy must be supported? How often is adaptation considered? Under what circumstances is adaptation cost-effective?

A system might, for example, modify itself to improve system response time, recover from a subsystem failure, or incorporate additional behavior during runtime. A system is open-adaptive if new application behaviors and adaptation plans can be introduced during runtime. A system is closed-adaptive if it is self-contained and not able to support the addition of new behaviors. A wide range of autonomy might be needed, from fully automatic, self-contained adaptation to human-in-the-loop. A wide range of policies can be used, from opportunistic, continuous adaptation to lazy, as-needed adaptation. The benefits gained from a change must outweigh the costs associated with making the change. Costs include the performance and memory overhead of monitoring system behavior, determining if a change would improve the system, and paying the associated costs of updating the system configuration. A wide range of strategies can be used, from continuous, precise, recent observations to sampled, approximate, historical observations.

II. RELATED WORK

Many researchers contributed towards dynamic self-adaptive software architectures. For instance architecture-based solutions [7], [24], [29] and self-healing systems [35] were explored by characterizing the style requirements of systems. The following sub sections provide more details of the review.

 ISSN: 2454-4248
641 – 649

Adaptation in Distributed Environment

Self-organizing systems that have provision for self-managing units that work with a common objective based on the software architecture [23].They are based on the concept known as architectural formalism of Darwin [33]. There are many components in the self-organizing system. Each one takes care of its own adaptation which is part of the whole system. To achieve this, each component remembers the architecture model. It can get rid of single point of failure besides having distributed control, consistent model. However, it causes significant overhead with respect to performance and needs an algorithm for global configuration. In our work we could overcome this problem by supporting global reorganizing and trade off between local and distributed controls while making adaption decisions.

Dynamic Architectures

Formal dynamic architectures can achieve self-adaptation. Towards this end many approaches came into existence. In [49] proposed a high level language to describe an architecture formally in such a way that it can reconfigure when changes are made to the components in the system and interconnections. The K-Component model proposed in [15] addresses two issues of the dynamic architectures. They are safety of evolution and integrity. They also took care of meta-models required for graph transformations. Many researchers focused on the programs called adaptation contracts to build applications that are self-adaptive [31]. Darwin is an example for ADL and used for describing distributed systems with self-adaptive architecture including operational semantics that take care of runtime dynamics of the system and reconfigurations [33]. Since the components and their organizations can change at runtime there is mechanism to analyze changes. Thus Darwin can help to be used as general purpose configuration language in distributed environment. Other known examples for ADL are PiLar [12] and ArchWare [37]. These are used to model architectural layers that can separate concerns and improve performance. These approaches are based on the reflective technologies for dynamic evolution. However, they assume that the implementations of systems are based on the formal architectural descriptions. Our approach in this paper decouples target system and external mechanisms for improving the adaption features.

Self-Adaption with Quality Attributes

Many architecture based solutions came into existence for self-adaptation. They are based on the quality attributes identified. For instance, they focused on performance as explored in [6], [27], and [32]. Survivability is another quality attributes used [50]. There are some researchers who focused on the architectural styles while building self-adaptive architectures [26], [38]. The style-based architectures had formal specifications that reflect different styles in self-adaptation. Many works in the literature are close to our work in this paper. They include the work done by UCI research group [14] and the work of Sztajnberg [47]. An architecture based solution for dynamic adaption at runtime was explored as an extension to [38]. This was achieved using planning loop, execution loop and monitoring. Its focus was on the self-adaption of C2-style systems. Merging and architectural differencing techniques were used in the implementation. Style-neutral ADL was explored y Sztajnberg and Loques besides many other aspects such as architectural contracts, and architectural reconfigurations. The model had support for formal verification but lacked in automated adaption of multiple objectives. In this paper, our approach addresses this problem. Rainbow framework [51] tried to provide far better solution for dynamic self-adaptive software architecture. However, it can be further improved with two additional modules as done by us in this paper. The modules are known as QoS analysers and Knowledge Mining. These modules can improve the performance of the architecture.

The approaches found in the literature have certain limitations and issues that need to be resolved. Many were addressed by Rainbow framework [51] such as exception handling and balance between local and global perspectives. It also focuses on quantity of adaption with many building blocks to achieve self-adaption. Customizable elements and reusable infrastructures are the two good features of Rainbow. However, the Rainbow framework has certain limitations as described here. Utility based theory was used for best adaptation path under uncertainty. The utility based frameworks are not fully quality-aware. Quality of Service (QoS) analyzers can be built to continuously monitor for improvement opportunities. The historical information usage is not sophisticated in the existing frameworks. It can be improved with state-of-the-art data mining techniques for improving decision making. All these issues are overcome in this paper with required modules incorporated into the framework.

III. PROPOSED SELF-ADAPTIVE DYNAMIC SOFTWARE ARCHITECTURE

In this section we describe the proposed self-adaptive dynamic software architecture. A self-adaptive software architecture can cater to the dynamic needs of the software at run time. It can adapt to runtime situations. Select adaptation needs to work for different kinds of systems and quality requirements. The adaptation is to be made with explicit operations that are chosen at run time. Such architecture should provide an integrated solution that saves time and effort of engineers as it can adapt to situations

 ISSN: 2454-4248
641 – 649

without the need for writing code and update the software explicitly. Our framework satisfies these requirements by supporting many mechanisms that lead to dynamic self-adaptation. Our architecture is known as Enhanced Self Adaptive Dynamic Software Architecture (ESADSA). This work has been influenced by Rainbow framework [31]. Our architecture shown in Figure 1 extends rainbow framework with two additional modules. They are known as Quality of Service (QoS) analyzers and History/Knowledge Miner.

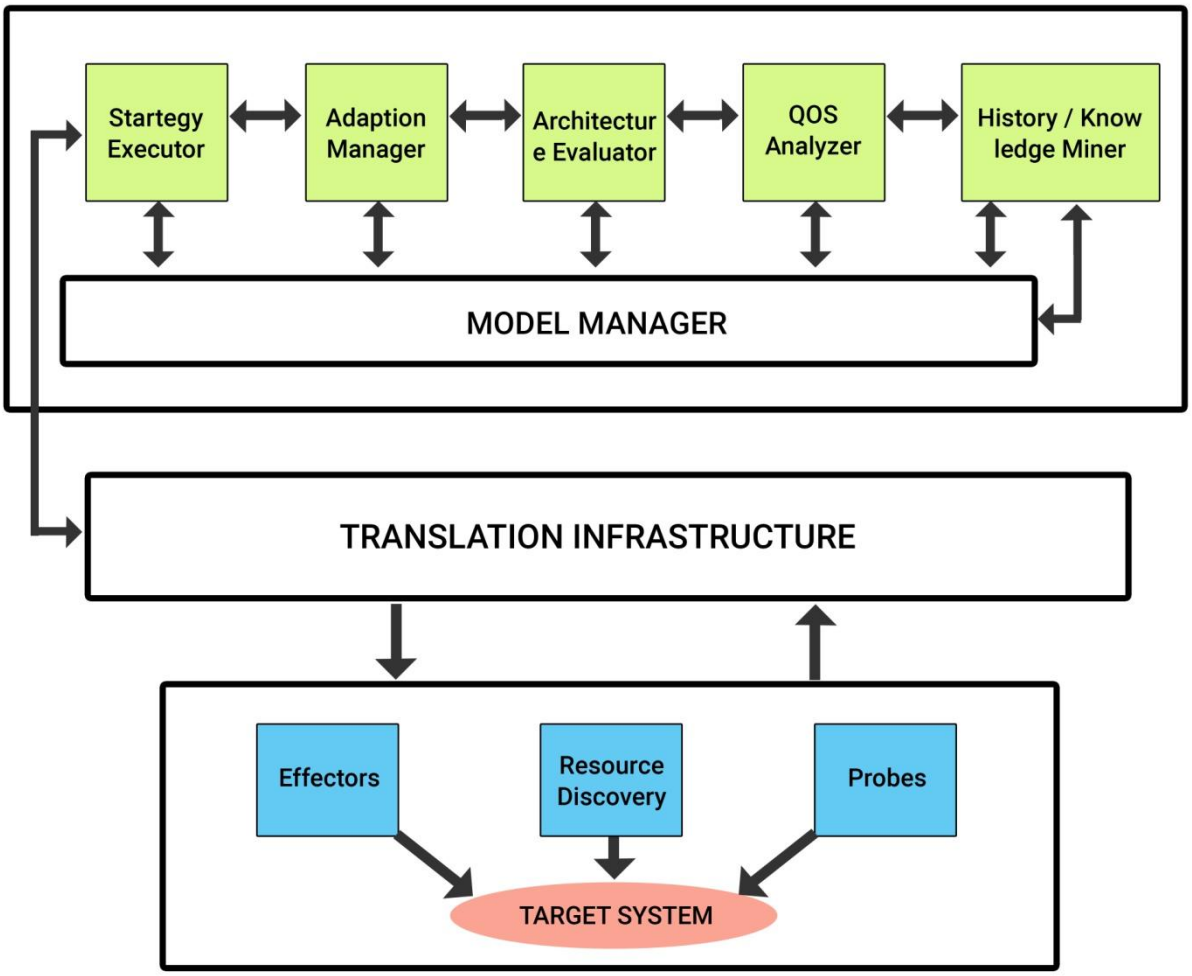

Figure 1 – The proposed framework named ESADSA

The framework has two layers known as architecture layer and system layer. Broadly, architecture layer represents self-adaptive mechanism which is made up of many components that work in tandem with each other. The system layer represents the target system and the plumbing components that are used to realize self-adaptation. There are two mechanisms for monitoring the target system. They are known as probes and gauges. The observations are reported to model manager. The architecture evaluator component is responsible to evaluate the model when model gets updates. It checks architectural constraints within acceptable range. When evaluation concludes that there is a problem in the system, the evaluation management invokes adaptation manager. The adaptation manager is responsible to initiate adaptation process and select a suitable adaptation strategy. Then the strategy executor comes into picture in order to execute the strategy on the runtime system. This is achieved through system level effectors that ensure realization of changes to the target system through self-adaptation.

There are three important aspects that are used to realize the software architecture. They are known as software architecture, control theory, and utility theory. Self-adaptation is possible with proposed software architecture that makes it cost-effective. Control theory and mechanisms ensure smooth adaptation. Utility theory helps in finding best strategy in self adaptation. Moreover the proposed architecture has two components for optimization. These components are known as QoS analyzers and knowledge miner.

Translation and Monitoring

 ISSN: 2454-4248
641 – 649

The translation infrastructure provided in architecture takes care of monitoring and action. This will help in bridging the gap between the target system and architecture layer. It has monitoring mechanisms like probes and gauges. These mechanisms get system states and update the model from time to time. The probe is responsible to measure target system while the gauge is responsible to interpret the measure identified. The probing is done on the attributes such as process run time and CPU load.

The solution given in [51] has certain limitations. They are described here. Utility based theory was used for best adaptation path under uncertainty. The utility based frameworks are not fully quality-aware. Quality of Service (QoS) analyzers can be built to continuously monitor for improvement opportunities. The historical information usage is not sophisticated in the existing frameworks. It can be improved with state-of-the-art data mining techniques for improving decision making. The modules incorporated in the framework for overcoming the drawbacks of [51] are as follows. These modules improve the performance of the architecture in making it robust and dynamic and self-adaptive software architecture.

QoS Analyzers

- These are the components in the architectural layer of the proposed framework.
- They take care of quality-aware analysis to exploit opportunities at runtime based on the QoS parameters like resource utilization, response time, etc.

Building & Mining Knowledge Base

A holistic historical information maintenance and analysis is made using state-of-the-art data mining techniques for making well informed decisions towards best adaptation path.

News.com Example with Self Adaptation

To demonstrate the usefulness of the proposed framework a hypothetical news web site by name News.com is taken as case study. This application is distributed in nature. It is n-tier application that has web based client. The tiers in it include client tier, web tier, business tier and data tier. It has a load balancer which will balance the load across the servers. In other words, the service is given from different servers. There are web based clients to the application where the clients make stateless requisitions without a session based approach. The requests are processed by different servers based on their availability.

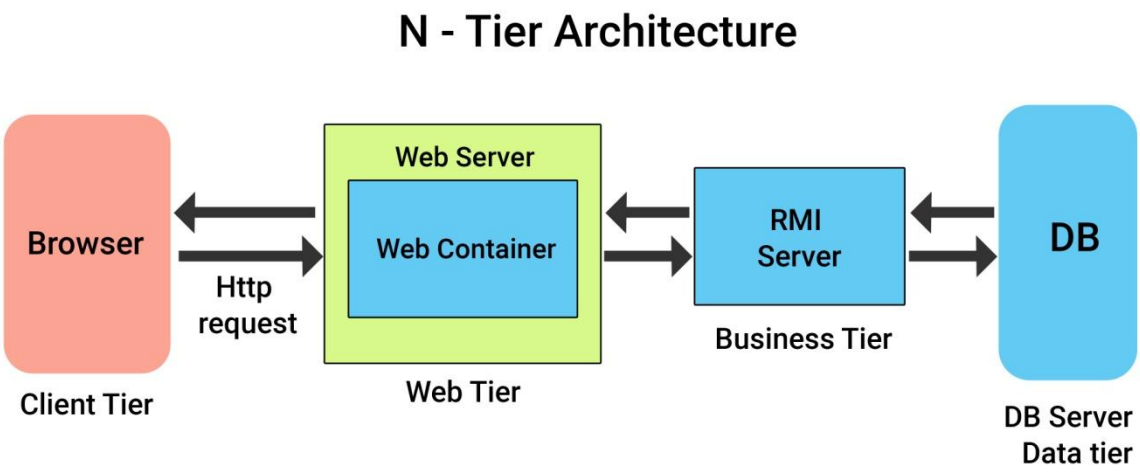

Figure 2 – Technical architecture of News.com

As can be seen in Figure 2, the browser is in the client tier using which users can make request for news. Web server is in web tier which invokes servlet that is responsible to process the request. However, the servlet cannot render news directly. It invokes a remote server program known as RMI server. The RMI server is in business tier where business rules are applied. The RMI server interacts with database which is the sources of the news. The tomcat server is a web server which has the following service architecture.

 ISSN: 2454-4248
641 – 649

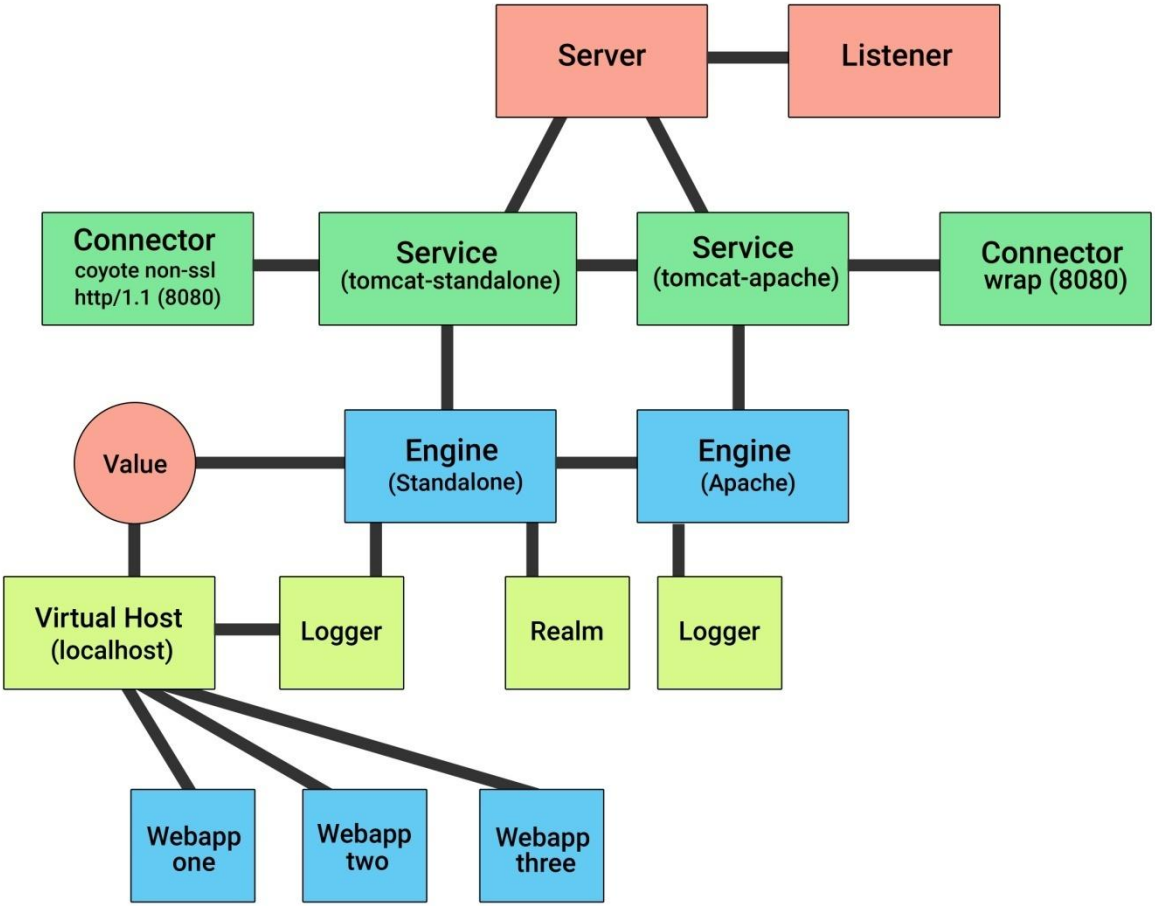

Figure 3 – Architecture of tomcat server

As can be seen in Figure 3, the tomcat server has support for different services and connectors. The connector may be coyote non-ssl or warp on 8080 port number. The services may be tomcat-standalone or tomcat-apache. Each service can make use of an engine that has required mechanism to have virtual hosts running different web applications. Web application is executed in tomcat server and servlets are executed by the servlet container or web container located in tomcat server.

The business model of News.com is to provide news content to its customers with good response time and with optimal cost at the server pool with operational budgets. However, the objectives of the News.com are not fulfilled when there are sudden spikes in the number of news requests from its clients due to certain popular events. This has resulted in unacceptable latencies and reduced customer satisfaction. The administrators of the application need to do certain configuration settings manually. They can either increase the server pool size or switch to different content mode. When there is heavy load the administrators make decisions and switch to text mode keeping cost and budgets of server pool in mind. The adaptation is done manually with the following objectives.

- The default content mode of the servers is multimedia
- When situation demands switch to textual model
- Increase the server pool size as and when required
- Decrease the server pool size as and when required

Performing these things manually is time taking and error prone. In this paper we strive to automate this process by implementing the proposed framework to make the News.com application self adaptive. Making well informed decisions automatically with tradeoffs among different objectives is a challenging task handled by the proposed framework.

In case of News.com the server response time is monitored and measured in the architecture model. This is done by model manager. The architecture evaluator ensures that the latency is not above given threshold. When it is not meeting the condition, the evaluator triggers the adaptation manager to make necessary steps in order to activate more servers or switch to text mode to reduce content quality. The system hooks are used by the strategy manager to make use of strategy that can handle the situation.

 ISSN: 2454-4248
641 – 649

News.com with Self Adaptation

The proposed framework is implemented using Java platform to realize self-adaptive dynamic architecture for News.com. The implementation of the framework is to ensure the self adaptation. It is independent of any application like News.com. It can be employed to any application that runs in the environment.

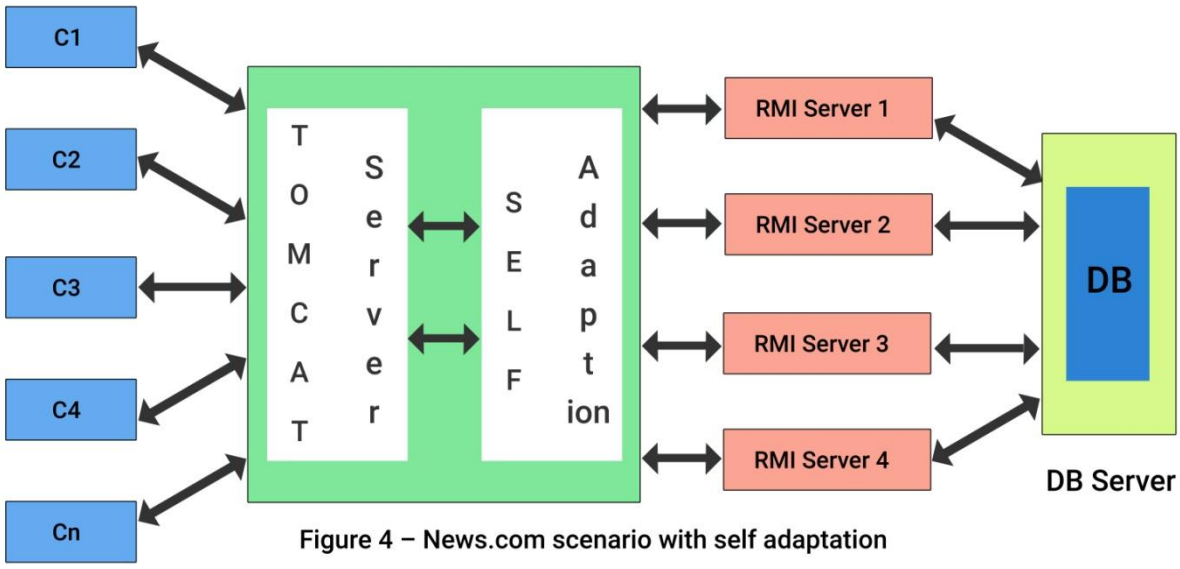

Figure 4 – News.com scenario with self adaptation

Figure 4 – News.com scenario with self adaptation

As shown in Figure 4, there are many clients making simultaneous requests to web server. The adaptation framework is there to monitor the requests and ensure that the servers are automatically configured. There are four important operations done here. They include increasing server pool size, decreasing server pool size, switching to text mode and switching to multimedia mode. The QoS analyzer proposed in the framework takes care of quality analysis. This knowledge coupled with the history miner's outcome can bring about best possible pat for self adaptation.

IV. EXPERIMENTAL RESULTS

Experiments are made with live News.com application built in the laboratory. It is executed in distributed environment using different machines connected to LAN. The number of requests is simulated and the latency and other observations are recorded.

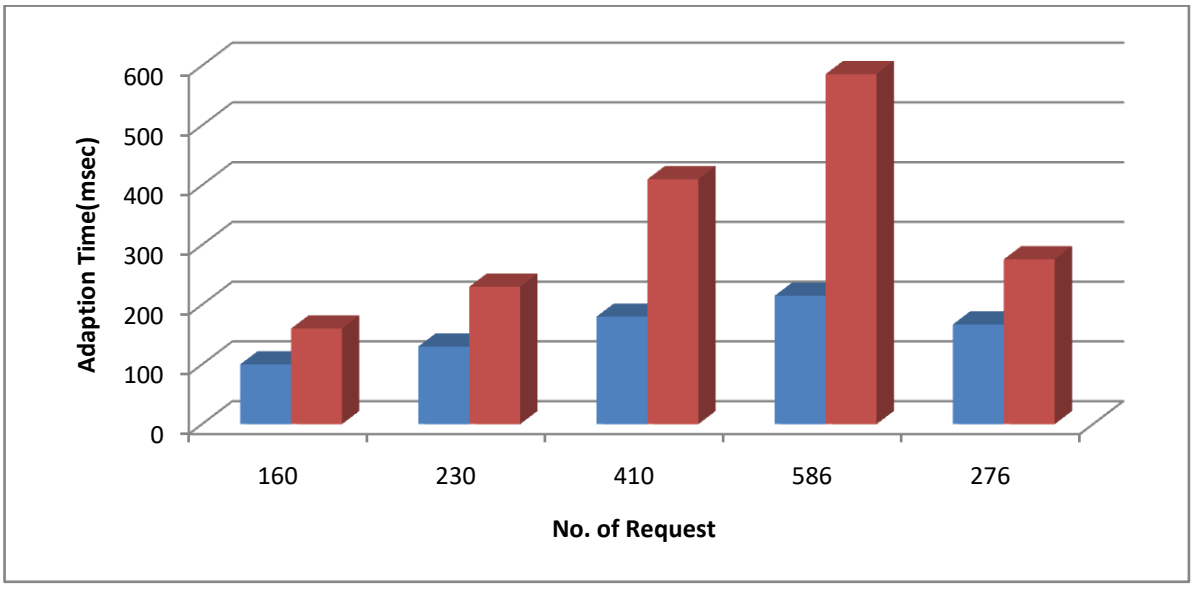

Figure 5 – The adaptation time vs. number of requests

 ISSN: 2454-4248
641 – 649

As can be seen in Figure 5, the trends in adaptation time when number of requests are increased or decreased are presented. The horizontal axis shows number of requests used in different experiments while the vertical axis shows the adaptation performance.

Figure 6 – Adaptation accuracy

The proposed system is evaluated with 100 experiments and found that it shows very negligible false positives. However, it is the proof that the system can be improved further in order to achieve 100% true positives.

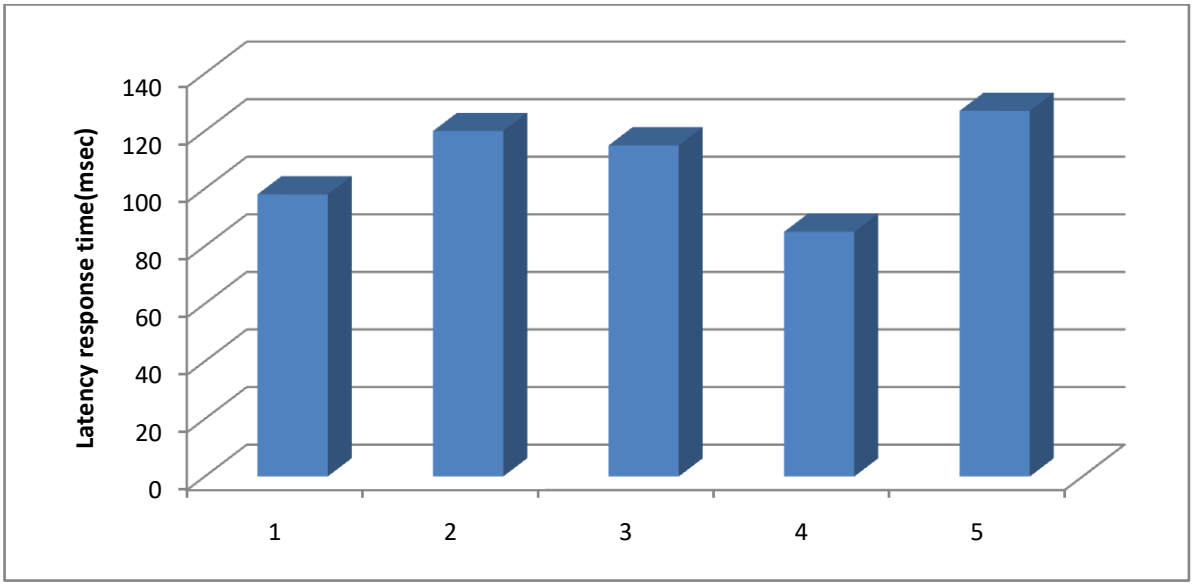

Figure 7 – Latency of random requests

The latency of different requests randomly chosen are observed and plotted in the graph shown in Figure 7. The latency time is presented in vertical axis while the number of random requests is shown in horizontal axis.

 ISSN: 2454-4248
641 – 649

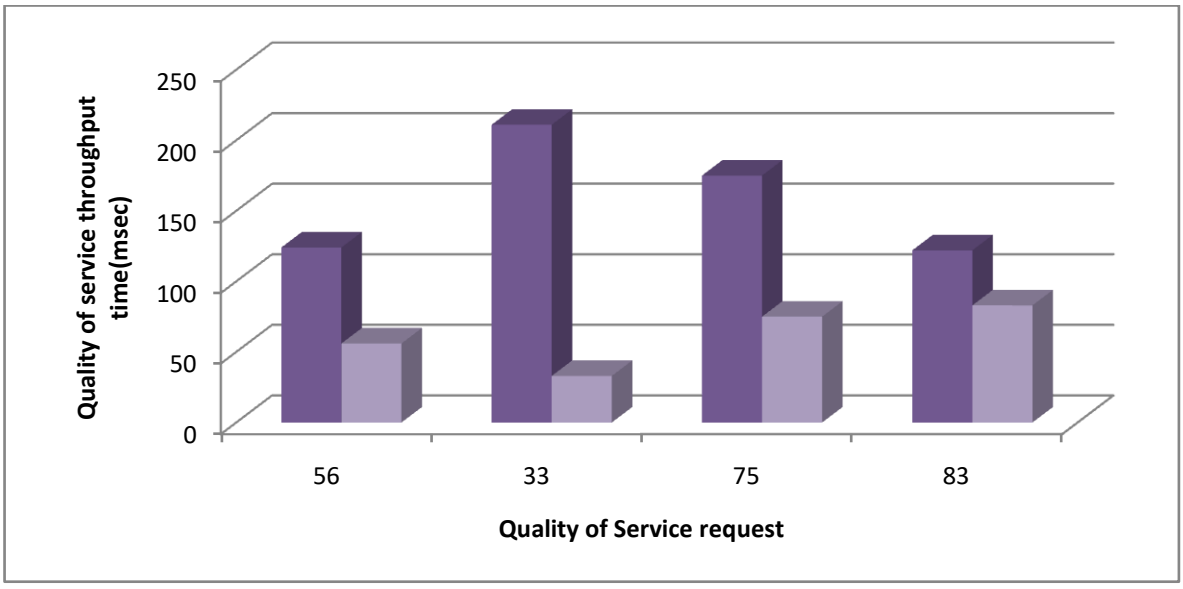

Figure 7 – Quality of service in terms of throughput

As can be seen in Figure 7, the quality of service is measured in terms of throughput. The response time is considered with different number of requests made to the News.com servers. The latency is recorded and presented.

V. CONCLUSIONS AND FUTURE WORK

Software systems have been evolving. The contemporary applications are able to drive businesses of multiple organizations that form as an integrated set of business. Applications in such environment need changes dynamically. The changes are to be incorporated in traditional approach by software engineers. However, this approach is costly and time consuming. To overcome this problem, many self-adaptive frameworks came into existence. Rainbow is one such architecture which enables dynamic self-adaptive software. However, the Rainbow architecture has specific limitations in terms of QoS analysis and knowledge mining in making decisions for adaptation. In this paper we proposed and implemented a self-adaptive dynamic software architecture named ESADSA which is based on a holistic approach with focus on QoS and knowledge mining for making expert decisions in adapting strategies as part of self-adaptation. We built a prototype application to demonstrate the proof of concept. Our results revealed that the proposed architecture significantly improves the self-adaptation performance. In future we focus on defining self-adaptive dynamic software architecture for cloud based systems.

REFERENCES

[1] Norha M. Villegas,Gabriel Tamura,Rubby Casallas. (2011). A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems. *ACM*, p.20-30.
[2] Ahmed Elkhodary,Naeem Esfahani,Sam Malek. (2010). FUSION: A Framework for Engineering Self-Tuning Self-Adaptive Software Systems. *ACM*, p.45-56.
[3] Andres J. Ramirez and Betty H.C. Cheng. (2010). Design Patterns for Developing Dynamically Adaptive Systems. *ACM*, p.901-1002.
[4] Pete Sawyer, Nelly Bencomo, Jon Whittle,Emmanuel Letier, Anthony Finkelstein. (2010). Requirements-Aware Systems. *IEEE*, p.90-101.
[5] Narges Khakpour, Saeed Jalili , Carolyn Talcott , Marjan Sirjani , MohammadReza Mousavie. (2012). Formal modeling of evolving self-adaptive systems. *Elsevier*. 78 , p.45-56.
[6] Jon Whittle Pete Sawyer Nelly Bencomo Betty H. C. Cheng Jean-Michel Bruel. (2010). RELAX: a language to address uncertainty in self-adaptive systems requirement. *Springer-Verlag London Limited*, p.12-19.
[7] Danny Weyns, M. Usman Iftikhar, Didac Gil de la Iglesia, Tanvir Ahmad. (2010). A Survey of Formal Methods in Self-Adaptive Systems.*ACM*, p.45-56.
[8] Daniel A. Menascé, Hassan Gomaa, Sam Malek, and João P. Sousa. (2011). SASSY: A Framework for Self-Architecting Service-Oriented Systems. *IEEE*, p.90-101.
[9] Naeem Esfahani,Ehsan Kouroshfar,Sam Malek. (2011). Taming Uncertainty in Self-Adaptive Software. *ACM*, p.45-56.
[10] Narges Khakpour , Saeed Jalil,Carolyn Talcott,Marjan Sirjani,MohammadReza Mousavi. (2010). PobSAM: Policy-based Managing of Actors in Self-Adaptive Systems. *Elsevier*. 263, p.20-30.
[11] Dhaminda B. Abeywickrama, Nicola Bicocchi, Franco Zambonelli. (2012). SOTA: Towards a General Model for Self-Adaptive Systems.*IEEE*, p.90-101.

 ISSN: 2454-4248
641 – 649

[12] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev and Erich Amrehn. (2014). Self-adaptive workload classification and forecasting for proactive resource provisioning. *John Wiley & Sons, Ltd*, p.901-1002.
[13] M. Usman Iftikhar Danny Weyns. (2012). A Case Study on Formal Verification of Self-Adaptive Behaviors in a Decentralized System.*ACM*, p.12-19.
[14] THOMAS VOGEL and HOLGER GIESE,. (2014). Model-Driven Engineering of Self-Adaptive Software with EUREMA. *ACM*, p.901-1002.
[15] Carlos Eduardo da Silva,Rogério de Lemos. (2011). Dynamic Plans for Integration Testing of Self-adaptive Software Systems. *ACM*, p.12-19.
[16] DANNY WEYNS,SAM MALEK,JESPER ANDERSSON. (2012). FORMS: Unifying Reference Model for Formal Specification of Distributed Self-Adaptive Systems. *ACM*, p.20-30.
[17] Daniel Sykes, Jeff Magee, Jeff Kramer. (2011). FlashMob: Distributed Adaptive Self-Assembly. *ACM*, p.901-1002.
[18] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw. (2010). Software Engineering for Self-Adaptive Systems: A Second Research Roadmap (Draft Version of May 20, 2011).ACM, p.90-101.
[19] Mark Harman, Edmund Burke, John A. Clark and Xin Yao. (2012). Dynamic Adaptive Search Based Software Engineering. *ACM*, p.45-56.
[20] Antonio Filieri, Carlo Ghezzi, Alberto Leva, Martina Maggio. (2011). Self-Adaptive Software Meets Control Theory: A Preliminary Approach Supporting Reliability Requirements. *IEEE*, p.20-30.
[21] Nelly Bencomo,Amel Belaggoun,Valerie Issarny. (2013). Dynamic Decision Networks for Decision-Making in Self-Adaptive Systems: A Case Study. *IEEE*, p.45-56.
[22] Antonio Filieri, Henry Hoffmann,Martina Maggio. (2014). Automated Design of Self-Adaptive Software with Control-Theoretical Formal Guarantees. *ACM*, p.12-19.
[23] J.L. Pastrana , E.Pimentel , M.Katrib. (2011). QoS-enabledandself-adaptiveconnectorsforWebServices compositionandcoordination.*Elsevier*. 37, p.901-1002.
[24] Markus Happe Enno Lu¨bbers Marco Platzner. (2013). A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking. *Springer-Verlag London Limited*, p.45-56.
[25] Francois Fouquet, Olivier Barais,Brice Morin Franck Fleurey. (2012). A Dynamic Component Model for Cyber Physical Systems. *ACM*, p.20-30.
[26] Yuankai Wu, Yijian Wu, Xin Peng, Wenyun Zhao. (2010). Implementing Self-Adaptive Software Architecture by Reflective Component Model and Dynamic AOP: A Case Study. *IEEE*, p.45-56.
[27] DANNY WEYNS,SAM MALEK,JESPER ANDERSSON. (2010). FORMS: a FOrmal Reference Model for Self-adaptation. *ACM*, p.20-30.
[28] Dimitrios Al. Alexandrou, Ioannis E. Skitsas, and Gregoris N. Mentzas. (2011). A Holistic Environment for the Design and Execution of Self-Adaptive Clinical Pathways. *IEEE*. 15 (1), p.90-101.
[29] Naeem Esfahani, Ahmed Elkhodary, and Sam Malek. (2013). A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems. *IEEE*. 39 (11), p.45-56.
[30] DANNY WEYNS,SAM MALEK,JESPER ANDERSSON. (2010). On Decentralized Self-Adaptation: Lessons from the Trenches and Challenges for the Future. *ACM*, p.90-101.