The Impact of Architectural Styles on Self Adaptive Systems Engineering

Yousef Abuseta, Khaled Swesi
Computer Science Department, Faculty of Science
Al-Jabal Al-Gharbi University
Zintan, Libya

***Abstract*— Despite the vital role of self adaptive systems in overcoming the challenges faced by today's software systems and the available approaches proposed so far, their design is still not conducted in a systematic manner. Amongst the research areas that self adaptive systems have been studied within is the software architecture and some approaches belonging to this area have suggested the use of architecture styles. Therefore, in this paper we investigate to what extent the architectural style can assist self adaptive system designers in identifying the quality properties to be monitored as well as the constraints imposed on them and the operators that represent the configuration changes. We also study the impact of the architectural style on the organisation of the feedback control loops used to augment software systems with self adaptability. We base our study on two architectural styles namely the client-server and pipe-filter styles.**

Keywords*— *self adaptive; architectural style; feedback control loop

I. INTRODUCTION

Increasingly, the engineering of modern software systems has become a challenging task due to the dynamic nature of the operational conditions in which these systems have to function. Such conditions include unstable resource availability, existence of errors that are hard to predict and changing user requirements [1]. These challenges have motivated the adoption of self adaptive systems. A self-adaptive system is a system that is able to change its structure or behavior at run-time in response to the execution context variations and according to adaptation engine decisions [2].

A large body of research has been carried out for engineering self adaptive systems within many areas. Amongst these research areas is the software architecture and some approaches [3] belonging to this area have suggested the use of architecture styles to facilitate the process of choosing what properties (local or global) to monitor as well as the concepts that exist in the system under consideration and the operators that represent the configuration changes.

In this paper we investigate the advantages of employing the architectural styles for the engineering of self adaptive systems. We also study the impact of the architectural style on the organisation of the feedback control loops. We also propose a set of architectural style based design patterns for designing self adaptive systems to assist the designers in designing such systems.

The rest of the paper is organized as follows. Section II serves as a background on related concepts and approaches to the work of this paper. Section III reviews some related work on self adaptive systems. Section IV presents the architectural styles for self adaptive systems engineering . The paper is concluded in section V with some outlined directions for future work.

II. BACKGROUND

A. Architectural Styles

The software architecture is a an abstract representation of the system as a composition of computational elements and their interconnections [4]. An architecture model provides a global perspective on the system and exposes the important system-level behaviors and properties such as the response time and throughput allowing the analysis of such quality of service properties. Also, the architecture model can be associated with a set of constraints that governs the architecture [5].

An architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of *constraints* on how they can be combined [6].

Architectural styles [7] include *client-server*, *pipe-filter*, *blackboard*, *etc.* The choice of the appropriate style for the target system is crucial for architecting self adaptive systems since, given some quality objectives, each style may guide the choice of system properties to monitor, help identify strategic points for system observation, and suggest possible adaptations [3][8].

B. Feedback control loops

Feedback loops provide the generic mechanism for self-adaptation [9]. A feedback loop is a control loop where the output of the controlled system is fed back to the input. It allows therefore to adjust operations according to differences between the actual output and the desired output. In other

words, feedback control loops are entities that observe the system and initiate adaption. A feedback loop typically involves four key activities: collect, analyze, decide, and act [10]. Sensors collect data from the running system and its environment which represents its current state. The collected data are then aggregated and saved for future reference to construct a model of past and current states. The data is then analyzed to infer trends and identify symptoms. The planning activity then takes place and attempts to predict the future and prepare change plan to act on the running system through a set of effectors or actuators. [9] .

C. *Managed and Managing Systems*

Self adaptation capabilities can be introduced to the software system either internally or externally [11]. In the internal approach, the adaptation logic (managing system) is intertwined with the core application (managed system) which may take the form of the exception handling. In this case, the adaptation engine is system dependent and thus difficult to maintain, evolve, and reuse. In contrast, in the external approach, the concerns of the adaptation logic are separated from the core application. Most of the existing approaches adopt the external approach since it enables the realization of some important software qualities such as the reusability and modifiability. The IBM architecture blueprint [12] is an example of this approach which is shown in Fig.1.

As depicted in Figure.1, the managing system consists of four main activities: monitor, analyze, plan and execute. These activities share a knowledge base component which contains information about the system state as well as the policy engine that controls the system functioning. A set of sensors is used to collect the important data to the adaptation process and send them to the monitor for further processing while a set of effectors is used to apply the corrective changes stated in the plan.

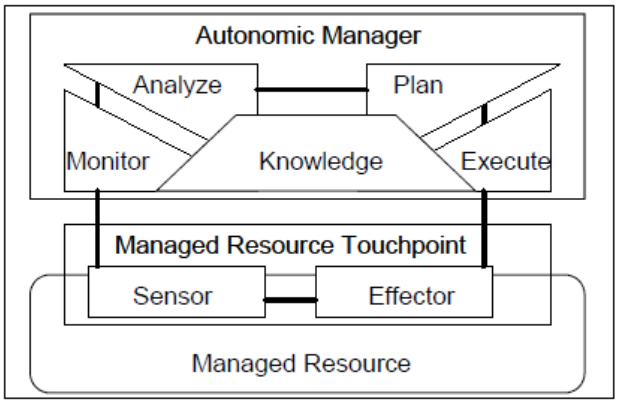

Fig. 1. IBM architecture blueprint [13].

III. RELATED WORK

Several approaches have been proposed to address the design of self adaptive software systems. These approaches can be classified to requirements engineering [14], software architecture [3][8][15], middleware [16], and component-based development [17]. In [3], Garlan et al proposed the Rainbow framework which provides general, supporting mechanisms for self-adaptation which can be customized for different classes of systems. It provides a language, called Stitch, to represent the adaptation knowledge using high-level adaptation concepts of strategies, tactics, and operators. The implementation of Rainbow is based on one large control loop that is in charge of all activities related to the self adaptability issue for the whole system under consideration.

Ramirez and Cheng [18] describe a set of design patterns at the software design level to facilitate the construction of a self-adaptive software system. They extended the pattern template used by Gamma et al. [19] for describing design patterns with Behavioral and Constraints fields.

Gomaa et al [20] proposed several patterns for dynamically reconfiguring specific types of software architectures at run time. In particular, they extended the concepts of dynamic change management introduced by Kramer and Magee [21] by introducing four design patterns to specify the behavior required to dynamically reconfigure master/slave, centralized, server/client, and decentralized architectures.

In [22], a set of design patterns is introduced by the authors to model the various interactions between MAPE loops in decentralized control of self adaptive software systems. This work can be used as a basis for understanding different patterns of decentralized control by software engineers of self-adaptive systems.

In [23], we have proposed and devised a set of design patterns for modeling the possible different interactions between the components in one feedback control loop as well as components from different control loops.

IV. ARCHITECTURAL STYLES FOR SAS ENGINEERING

In this section we discuss and investigate the consequences of adopting architectural styles when engineering self adaptive systems. Also, we study the impact of an architectural style on the organization and layout of the feedback control loop of IBM architecture blueprint. Our discussion is based on, two different architectural styles namely the client-server and pipe-filter styles.

A. *Client-server architecture style*

In the client-server architectural style [6][7] , the system typically consists of three fundamental concepts namely the client, server and the network connection. A request-reply connection binds a client to a specific server where the client requests a particular resource residing on the server and the server replies to this request. A server is set up to receive

requests from many clients simultaneously. A server can be overloaded with requests which causes it to be either too slow or completely unresponsive. Thus, an application providing certain services must be able to run on multiple servers to accept and accommodate an ever increasing number of users. Scalability is therefore very important issue here and must be dealt with. Also, a server can be down at anytime which affects the service provided by the application running on the server (or servers). Therefore, availability is an issue here and must be addressed too. The scalability issue is addressed by the load balancing technique in which service requests are distributed (via server load balancer) across a set of identical servers. Availability is solved through the load balancing technique too. However, unless there is a mechanism for monitoring the health of the servers and then dispatching client requests accordingly, the load balancing can cause problems rather than solving them. A typical example of this kind of load balancer is the DNS round-robin. Fig. 2 illustrates a typical client-server architecture with load balancing in place.

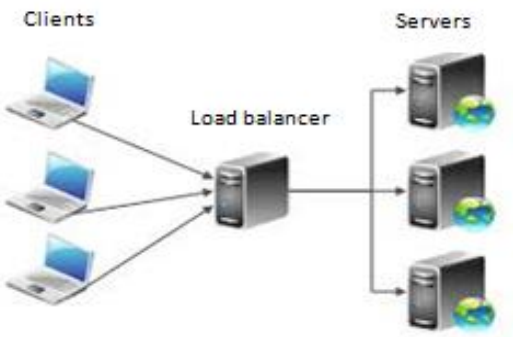

Fig. 2. Client-Server Architecture with load balancing.

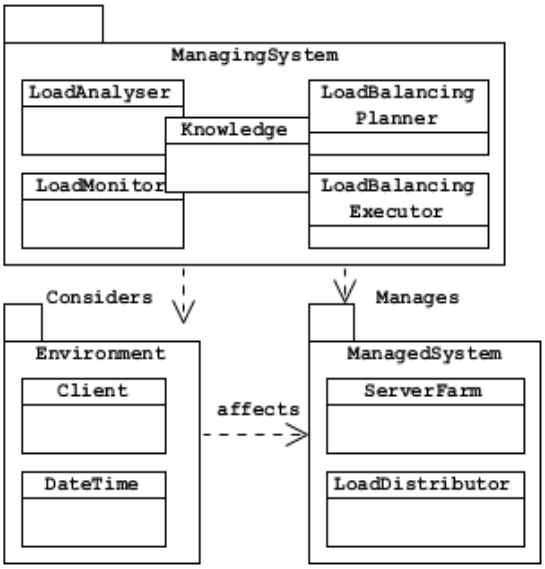

Fig. 3. Client-server style based self adaptive system.

B. Analysis of Client-Server based Self-adaptive systems

The client-server architectural style contains three primary concepts, namely the *client*, *server* and *network connection*. In terms of the software architecture, this style has two component types and one connector. These three concepts comprise the *managed system* that has to be monitored and made self adaptive. The load balancing is a mechanism to achieve scalability and availability and thus it represents actions that belong to the adaptation logic or the *managing system*.

1) Candidate properties for monitoring: The properties that can be identified here along with the components that they may apply to are as follows:

- *load and throughput* properties which are associated with the server component. Load expresses how many processes are waiting in the queue to access the server processor while throughput measures the amount of messages that a server processes during a specific time interval.
- *latency property* is associated with the *client* component since latency is measured from the client machine which is the time taken from sending a request to receiving a reply.
- *cost property* is associated with the Server component.

2) Candidate operators for adaptation actions: The operators, which represent the adaptation actions taken by the managing system, that can be identified here are described as follows:

- *Add server*: this operation causes the addition of a new server to the server farm as a corrective action to the sever high load.
- *Remove server*: this operation causes the removal of a server from the server farm either because of a low load at specific time or as a response to an unresponsive server.
- *Redirect request to different server*: this action is taken when one server is over loaded with client requests so subsequent requested is sent to less loaded server.

3) The Managing System for client-server Architecture:

When managing a client-server based software system, the load balancing technique largely contributes to solving many server related problems including the response time, throughput and scalability. Therefore, the managing system is necessarily composed of components concerned with the load balancing mechanism. Such components include load monitor, load analyzer, load balancing planner and load balancing executor. A knowledge component is used here by these components to achieve their tasks. Fig. 3 depicts the organization of the client-server style based self adaptive system (managing and managed systems).

A description of these components is presented as follows:

- *load monitor*: its main task is to monitor the server load using a set of sensors and report it to the load analyzer for analysis.
- *load analyzer*: once it has received the information from the load monitor, it analyzes it against some predefined threshold and goals. If a violation has been detected, an alert signal is sent to the load balancing planner with the appropriate and necessary information.
- *load balancing planner*: the main task of this component is to prepare the plan that contains the strategy, along with its actions, that is composed as a response to some detected problems.
- *load balancing executor*: the main task of this component is to dispatch the actions supplied by the load balancing planner and apply it to the managed system (the server farm).
- *Knowledge base*: this component contains the data and information necessary to manage the system in question such as the values of the monitored properties (server load, server status) which represent the ***current system state***. It also contains the **policy engine** that consists of a set of actions where each *action* is triggered as a response to an *event* provided a specific *condition* is held.
- Managed system: it represents a set of servers that is in charge of providing the services to the clients. Also, we consider the load distributor as part of the managed system since it is very likely to be in an overload state as well.
- The Environment: since the environment is defined as any external actor that affects the system in some way [Brun], we here believe that the client fits this definition and therefore better off modeled as part of the environment rather than of the managed system. Also, the date or time can affect the system operating and thus be considered as part of the environment. This is due to the fact that the system can be on high or low demand in some specific period of time.

However, as it is almost always the case that a self adaptation in distributed systems is likely to be realized using more than one MAPE control loop, addressing the interaction between these control loops is a must. A treatment of some of the existing interaction patterns is introduced in [22]. In our approach, we adopt the master-slave pattern [22] in which the adaptation logic takes a hierarchical relationship between one master component who is in charge of the analysis and planning of the adaptations and between a set of slave components whose responsibilities are to monitor and execute the adaptation actions. Therefore, the organization of the client-server style based self adaptive system depicted in Figure 3 becomes as shown in Fig. 4. Our design decision was driven by the desire of decreasing the monitoring overhead so the slave component is less loaded with tasks and monitoring is conducted locally.

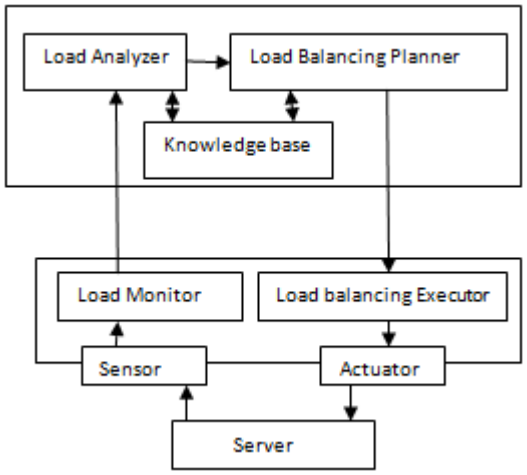

Fig. 4. Proposed client-server style based self adaptive system.

As depicted in Fig. 4, a set of interactions between the different components can be identified which is described as follows:

- *Sensor* to *monitor* interaction: here a sensor senses its own managed server for a particular property (e.g. CPU utility or number of current connections) and sends it to the monitor. This interaction can either be accomplished using the pull or push approach. Also, the interaction can be either event-triggered or time triggered.
- Monitor to analyzer interaction: this interaction involves a set of monitors reporting event information about the possible threshold violation either of the CPU utility (an indicator of high load on the server) or the maximum number of current connections (a possible indicator of high load).
- Analyzer to planner interaction: in this interaction the analyzer notifies the planner of any required adaptations once the current system (server farm) state is drifted away from the desired or accepted state.
- Planner to executor interaction: upon receiving a notification of required adaptation from the analyzer, the planner creates a strategy with the appropriate actions and send it to the executor to dispatch these actions. The planner checks policy engine that resides in the knowledge base component to accomplish its task.
- Executor to actuator interaction: required change/adaptation actions are sent from the executor to the actuator.

C. Pipe-filter architectural style

The Pipe and filter architecture is a type of data flow architecture where the flow is driven by data. In such a style, the system is based on two fundamental concepts: filters and pipes [24]. A filter is a component that reads a stream of data as input and produces a stream of data as output. A pipe is a connector that transmits the output of one filter and feed it as input to another filter. Among the important invariants of the style is the condition that filters must be independent entities: in particular, they should not share state with other filters. Another important invariant is that filters do not know the identity of their upstream and downstream filters. Thus, filters may not identify the components on the ends of their pipes [7] [24]. A common specialization of this style is the pipeline architecture where the topology is restricted to a linear sequence of filters. Fig. 5 illustrates a typical pipe-filter architecture.

D. Analysis of pipe-filter based Self-adaptive system

As stated earlier, a pipe-filter architectural style contains two primary concepts, namely the *pipe and filter*. In terms of the software architecture, this style has one component type and one connector. These two concepts comprise the *managed system* that has to be monitored and made self adaptive. The throughput and security are two important system properties to the software systems complying with the pipe-filter style. Therefore, the *managing system* contains the adaptation logic that is concerned with controlling and adjusting the overall system throughput.

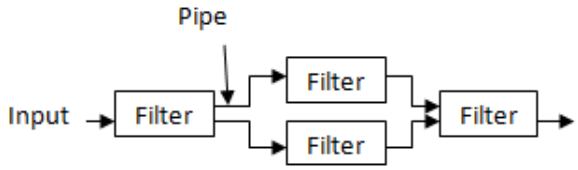

Fig. 5. Possible organization of pipe-filter architecture style.

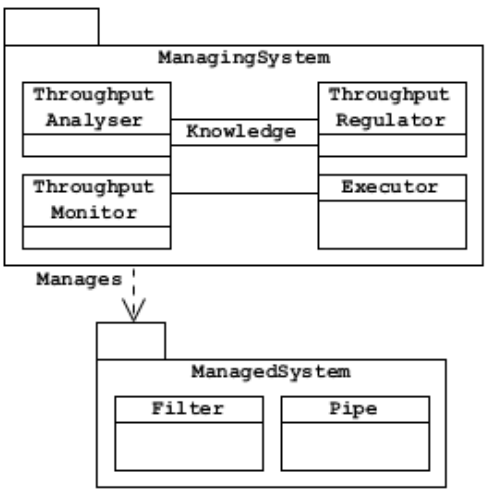

Fig. 6. Pipe-filter style based self adaptive system.

1) Candidate properties for monitoring: The properties that can be identified here along with the components that they may apply to are as follows:

- *Throughput* property which is associated with the the whole set of filters that is taken part in the system functionality. Throughput measures the amount of messages that the filters process during a specific period of time.
- *Transfer rate* property which is associated with the *pipe* component. Such a property is measured by the time taken from transferring some output from one filter to another .
- *Processing time* property which is associated with the filter component. Such a property is measured by the time taken by a filter to accomplish the task or transaction assigned to it.

2) Candidate operators for adaptation actions: The operators, which represent the adaptation actions taken by the managing system, that can be identified here are described as follows:

- *Replace filter*: this operation causes the replacement of one filter with another in order to improve the overall system throughput.
- *Encrypt data*: this action is taken when the security is of high priority so data is encrypted prior to sending them to the target filter.
- *Replace pipe*: this operation causes the replacement of one pipe with another in order to improve the overall system transfer rate.
- *Provide alternative paths:* this action is taken to increase the system robustness and reliability.

3) The Managing System for pipe-filterr Architecture:

When managing a pipe-filter based software system, two issues are of great importance to the managing system: the processing time of the filter and transfer rate of the pipe. Both contribute to the improvement of the overall system throughput. Therefore, the managing system is necessarily composed of components concerned with managing and regulating the throughput property of the system in question.. Such components include processing time/ transfer rate monitor, processing time/ transfer rate analyzer, processing time/ transfer rate planner and load balancing executor. A knowledge component is used here by these components to achieve their tasks. Fig. 6 depicts the organization of the pipe-filter style based self adaptive system (managing and managed systems).

A description of these components is presented as follows:

- *Throughput monitor*: its main purpose is to monitor the system throughput using a set of sensors and report it to the throughput analyzer for analysis.
- *Throughput analyzer:* once it has received the information from the throughput monitor, it analyzes it against some predefined threshold and goals. If a violation has been detected, an alert signal is sent to the throughput regulator with the appropriate and necessary information.
- *Throughput regulator:* the main task of this component is to prepare the strategy, which consists of the corrective actions, that is composed as a response to some detected undesirable states regarding the throughput system property.
- *Executor:* the main task of this component is to dispatch the actions supplied by the *Throughput regulator* and apply it to the managed system (the filters and pipes).
- *Knowledge base:* this component contains the data and information necessary to manage the system in question such as the values of the monitored properties (filter processing time, pipe transfer rate) which represent the ***current system state***. It also contains the **policy engine** that consists of a set of actions where each *action* is triggered as a response to an *event* provided a specific *condition* is held.
- *Managed system:* it represents a set of filters and pipes. The filters are monitored for their processing times while the pipes are monitored for their transfer rates.

As stated earlier, the self adaptation is usually achieved using multiple feedback control loops and therefore this issue must be taken into consideration when designing self adaptive systems. Accordingly, pipe-filter based self adaptive system presented in Fig. 6 is refined and detailed as shown in Fig. 7.

V. Conclusion and Future work

In this paper we have investigated two main issues: (1) The effectiveness of designing a self adaptive system around the idea of the architectural styles and (2) The impact of the architectural style on the organisation of the feedback control loops used to augment software systems with self adaptability. Our discussion was based on the client-server architectural style and pipe-filter architectural styles. In client-server architectural style, the load balancing mechanism was applied to achieve some important quality of services (QoS) properties such as the system performance and availability. Accordingly, the MAPE components were termed as load monitor, load analyzer, load balancing planner and load balancing executor. In the pipe-filter architectural style, the throughput issue was identified as the highest important property to manage and regulate and therefore the components of the managing system of this style are designed and modeled around this property where the filter processing time and pipe transfer rate parameters are used to achieve this task.

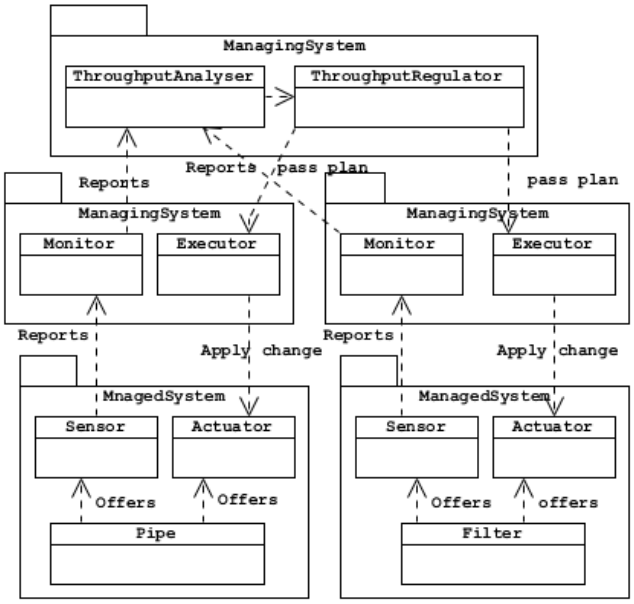

Fig. 7. Proposed pipe-filter based self adaptive system.

We have found that using the architectural styles can help in exposing the system properties that have to be monitored and kept at some desirable range. Also, the operators which represent the adaptation actions that are taken in response to violations on some thresholds can be easily identified. However, these advantages can only facilitate the SAS design at the system level where some quality of services, such as scalability and availability, are of great importance. Therefore, more work is needed to include the self adaptive capabilities related to the core services (usually at the component or component composite level) of the system in question.

Then, we have devised a design pattern that can be used by the system designer of self adaptive systems adopting the client service style to design such systems in a systematic way. We adopted the master-slave pattern in which the adaptation logic takes a hierarchical relationship between one master component who is in charge of the analysis and planning of the adaptations and between a set of slave components whose responsibilities are to monitor and execute the adaptation actions. We also proposed s design pattern for the pipe-filter based self adaptive systems where two properties (processing time and transfer rate) are monitored to keep the throughput at some desirable range. We have also used a hierarchical relationship here as in the client-server style where the adaptation analysis and planning are performed at a single central control loop.. However, we could have treated each component in this style (pipe and filter) as an autonomous entity and each one is augmented with its own control loop to keep the observed and regulated property (processing time or transfer rate) at a desirable range using a threshold.

A further work is required to address the following issues:

- More detailed design patterns for architectural style driven self adaptive systems modeling.
- A complete lifecycle for the development of self adaptive systems taken into account the architectural styles.
- Investigation and analysis of more architectural styles regarding the engineering of self adaptive systems
- The investigation of the benefit of integrating the architectural styles with the domain specific languages (DSLs).

ACKNOWLEDGMENT

The authors would like to thank their academic department for the precious support and encouragement received throughout the work on this paper.

REFERENCES

[1] D. Iglesia, "A Formal Approach for Designing Distributed Self-Adaptive Systems" , PhD Thesis, Linnaeus University 2014.

[2] P. Oreizy, M. Gorlick, and R. N. Taylor, "An architecture-based approach to self-adaptive software". IEEE Intelligent Systems and Their Applications, vol. 14, no. 3, pp. 54–62, 1999.

[3] D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste,"Rainbow: architecture-based self-adaptation with reusable infrastructure". IEEE Computor Society Press, vol. 37, no. 10, pp. 46–54, 2004.

[4] M. Shaw, and D. Garlan"Software Architecture: Perspectives on an Emerging Discipline". Prentice-Hall, 1996.

[5] D. Garlan, B. Schmerl, and S. Cheng, " Software Architecture-Based Self-Adaptation", Springer Science+Business Media, LLC 2009.

[6] D. Garlan, and M. Shaw. "An Introduction to Software Architecture". Tehnical Report CMU/SEI-94-TR-21. School of Computer Science, Carnegie Mellon University, 1994.

[7] P. Clements, "Documenting Software Architectures". Addison-Wesley, 2011.

[8] D. Garlan, S. Cheng, and B. Schmerl. "Increasing system dependability through architecture-based self-repair". In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems. LNCS, vol. 2677. Springer, Heidelberg, 2003.

[9] Y. Brun."Engineering Self-Adaptive System through Feedback Loops," in Software Engineering for Self Adaptive Systems", Springer-Verlag Berlin Heidelberg 2009, Cheng B. et al. (Eds), pp 48-70, 2009.

[10] S. Dobson, "A survey of autonomic communications". ACM Transactions Autonomous Adaptive Systems (TAAS) 1(2), 223–259 (2006).

[11] M. Salehie and L. Tahvildari , "Self-adaptive software: landscape and research challenges," ACM Transactions on Autonomous and Adaptive Systems, vol. 4, no. 2, article 14, 2009.

[12] IBM. "An architectural blueprint for autonomic computing. IBM", 2005.

[13] B. Jacob, R. Lanyon-Hogg, D. Nadgir, and A. Yassin, "A Practical Guide to the IBM Autonomic Computing Toolkit". IBM 2004.

[14] G. Brown, B. Cheng, H. Goldsby, and J. Zhang, "Goal-oriented specification of adaptation requirements engineering in adaptive systems". In: ACM 2006 International Workshop on Self-Adaptation and Self-Managing Systems (SEAMS 2006), Shanghai, China, pp. 23–29, 2006.

[15] U. Richter, M. Mnif, J. Branke, C. Muller-Schloer, and H. Schmeck, "Towards a generic observer/controller architecture for organic computing". In: Hochberger, C., Liskowsky, R. (eds.) INFORMATIK 2006: Informatik f¨ur Menschen. GI-Edition – Lecture Notes in Informatics, vol. P-93, pp. 112–119 , 2006.

[16] H. Liu, and M. Parashar, " Accord: a programming framework for autonomic applications". IEEE Transactions on Systems, Man, and Cybernetics. vol. 36, no. 3, pp. 341–352, 2006.

[17] C. Peper, and D. Schneider,"Component engineering for adaptive ad-hoc systems". In: ACM 2008 International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). 2008.

[18] A. J. Ramirez and B. H. C. Cheng, "Design patterns for developing dynamically adaptive systems," in Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '10), pp. 49–58, ACM, USA, 2010.

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1995.

[20] H. Gomma, and M. Hussien, "Software Reconfiguration Patterns for Dynamic Evolution of Software Architectures", IEEE Conference on Software Architecture (WICSA'04), 2004.

[21] J. Kramer and J. Magee,"The Evolving Philosophers Problem: Dynamic Change Management", IEEE Transactions on Softwr Eng, Vol. 16, No. 11, Nov 1990.

[22] D. Weyns, "On patterns for decentralized control in self-adaptive systems". In R. de Lemos, H. Giese, H. A. Muller, and M. Shaw, editors, ¨ Software Engineering for Self-Adaptive Systems, volume 7475 of Lecture Notes in Computer Science, pages 76–107. Springer, 2012.

[23] Y. Abuseta, K. Swesi, "Design Patterns for Self Adaptive Systems Engineering". International Journal of Software Engineering & Applications (IJSEA), Vol.6, No.4, July 2015.

[24] K. Qian, "Software Architecture and Design Illuminited". Jones and Bartlett Publishers, 2010.