Research Article **ISSN: 2395 – 5597**

International Journal of Engineering and Robot Technology

Journal home page: www.ijerobot.com

METHODOLOGIES OF SELF-HEALING AND SYSTEM CONFRONTS (THE AUTONOMIC COMPUTING)

Raja Adeel Ahmed*[1], Shamasur Rehman[2], Naveed Anjum[2], Teklay Tezfazghi[3]

*[1]Computer Science Department, Eritrea Institute of Technology, Asmara, Eritrea, North East Africa.
[2]Mathematics Department, Eritrea Institute of Technology, Asmara, Eritrea, North East Africa.
[3]Computer Engineering Department, Eritrea Institute of Technology, Asmara, Eritrea, North East Africa.

ABSTRACT

Autonomic computing was introduced by IBM in 2001. Autonomic computing is a computer environment that can detect and adjust its system automatically to manage itself without the assistance of any human interaction. Autonomic computing is emerging as a significant new strategic and holistic approach to the design of complex distributed computer systems. It is inspired by the functioning of the human nervous system and is aimed at designing and building systems that are self-managing. Self-management is achieved through key aspects such as self-governing, self-adaptation, self-organization, self-optimization, self-configuration, self-diagnosis of faults, self-protection, self-healing, self-recovery, and autonomy. In this paper, we focus on the self-healing branch of the research and provide an overview of the current existing approaches. The paper is introduced by an outline of the origins of self-healing. Based on the principles of autonomic computing and self-adapting system research, we identify self-healing systems' fundamental principles. The extracted principles support our analysis of the collected approaches.

KEY WORDS

Autonomic computing, Self-adaptation, Self-organization, Self-optimization, Self-configuration, Self-healing and Self-recovery.

Author of correspondence:

Raja Adeel Ahmed,
Computer Science Department,
Eritrea Institute of Technology,
Asmara, Eritrea, North East Africa.

Email: adeelrajj@yahoo.com

INTRODUCTION

For decades, the advancement of technology and science has mirrored the increase of complexity in many computer environments. Recently, the advancement of computer technology and science has not been increasing at the same pace as complexity in computer environments. This unbalance in advancement creates a major obstacle. The major obstacle that concerns researchers is centered on complexity. However, as the scale and

complexity of these systems and applications grow, their development, configuration and management challenges are beginning to break current paradigms, overwhelm the capabilities of existing tools, methodologies, rapidly render the system, applications, brittle, unmanageable, and insecure[1]. As complexity increases, computer environments are being impacted with more failures and downtown. Most frequently cited outages included[2].

In Systems

Operational error, user error, third-party software error, internally developed software problem, inadequate change control, lack of automated processes.

In Networks

Performance overload, peak load problems, insufficient bandwidth.

In Database

Out of disk space, log file full, performance overload.

In Applications

- Application error, inadequate change control, operational error, non-automated application exceptions.
- Researchers became very concerned about the current epidemic of computer environments being destroyed due to complexity. For this purpose, researchers were faced with the task of finding an alternative approach to complexity. After exploring multiple methodologies, researchers finally developed a solution to the problem of complexity. The methodology researchers used to overcome the barrier of complexity is called autonomic computing.
- Autonomic computing is a computer environment that can detect and adjust its system automatically to heal itself without the assistance of any human interaction. Figure No.1 displays a typical procedure implemented in various IT organizations. Therefore, the IT industry was in need for a computer system that would foresee the users need and allow users to focus more on completing their work tasks and less on troubleshooting their computer system. Autonomic computing was conceived to lessen the spiraling demands for skilled IT resources, reduce complexity, to drive computing into a new era that may better exploit its potential to support higher order thinking and decision making. Implementing an autonomic computing system will help companies eliminate the increasing costs of restoring hardware and software failures. This methodology could help IT professionals develop more reliable and dependable systems within computer environments. Consequently, autonomic computing will effectively prevent downtimes and system failures. In addition to less downtimes and system failure, the production rate for computer environments controlled by autonomic systems will increase dramatically. For that reason, autonomic computing is emerging significantly in the IT industry.

Self-healing Principles

In this part we will describe the main principles of self-healing systems. It will help to understand the design decisions of the researched approaches and their underlying structures. Starting with a current definition of self-healing systems, we identify the important parts of a self-healing system to give a detailed insight into their purpose, composition, and functionality.

What is self-healing?

Self-Healing denotes the system ability to examine find, diagnose and react to system malfunctions. Self-healing components or applications must be able to observe system failures, evaluating constraints imposed by the outside, and to apply appropriate corrections. In order to automatically discover system malfunctions or possible futures failures, it is needful to know the expected system behavior. Autonomic systems must have knowledge about own behavior then they must have a knowledge in order to determine if the actual behavior is consistent and expected in relation of the environment. In new contexts or in different scenarios, new system behaviors can be observed

and the knowledge module must evolve with the environment[3].

Self-Healing systems basically endure a process in order to maintain satisfactory quality of service of the principal system during runtime in the presence of any fault. The first cycle is called the monitoring cycle. During the monitoring cycle, the systems monitor will inspect the computer environment for any improper conduct. After the monitor's inspections are complete, it will send the data gathered through current observations to the next stage. The second phase of the cycle is called error detection and diagnosis; if the diagnosis reports that there is no fault in the system then it will loop back to the monitor for more observations. If there is an error detected by the monitor, the error detection cycle will report it to the next stage of the cycle. The third stage of the cycle is known as analysis and selection of a repair operation. At this stage, the fault is analyzed and a method of repairing is determined at this part of the cycle. After the repair operation is determined, the report is passed onto the final phase of the cycle called execute repair and operation (self-repair). Any repairs that are needed are completed at this phase in the cycle. Once, the faulty areas are self-repaired the cycle begins all over again. Since this cycle is a closed loop, the process of self-healing environments will continuous heals itself as depicted in Figure No.2.

Included by IBM[4] self-healing is one of the main four properties defining an autonomic system, Ghosh[8] also provide a most recent definition of self-healing systems:

"…a self-healing system should recover from the abnormal (or "unhealthy") state and return to the normative ("healthy") state, and function as it was prior to disruption."

Pierce[5] stated that Fault-tolerant systems include stabilization techniques and replication strategies as essential methods for recovery. Thus, Ghosh[6] confess that self-healing systems in some cases are seen as subsidiary to fault-tolerant systems. Survivable systems handle malicious behavior by containing failing components and securing the "vital services" representing a minimal but functioning system configuration[7,8,9]. Normally, the focus of self-healing research is on recovery as an elaborate process. This comprises both, methods for stabilizing, replacing, securing and isolating, but more essentially, strategies to repair and prevent faults[6] identify the key aspect of self-healing systems as recovery oriented computing. This might also be a reason, why some of the researched approaches outline self-healing only as an enhanced recovery method (e.g.[10],[11]). Ganek and Corbi[7] further detail self-healing applications' operation mode as an organized process of detecting and dividing a faulty component, taking it off line, fixing the failed component, and reintroducing the fixed or replacement component into the system without any apparent disruption. For Ganek and Corbi[4] the objective of self-healing properties is to support system's reliability by minimizing the outages. Additionally, self-healing systems should be able to anticipate conflicts trying to prevent possible failures.

To summarize, the reason for enhancing a system with self-healing properties is to achieve *continuous availability*. Compensating the dynamics of a running system, self-healing techniques momentarily are in charge of the *maintenance of health*. *Enduring continuity* includes resilience against intended, necessary adaptations and unintentional, arbitrary behavior. Self-healing implementations work by *detecting* disruptions, *diagnosing* failure root cause and deriving a remedy, and *recovering* with a sound strategy. Additionally, to the accuracy of the essential sensor and actuator infrastructure, the success depends on timely detection of system misbehavior. This is only possible by continuously analyzing the sensed data as well as observing the results of necessary adaptation actions. The system design leads to a *control loop* similar assembly. An environment dependent and preferably adaptable set of *policies* support remedy decisions. Possible policies include simple sets of event dependent instructions but also extended artificial intelligence (AI) estimations supporting the resolution of previously unknown faults. A conspectus of the research on self-healing properties is given in Figure No.3. At the bottom, the origins of the self-healing ideas are illustrated. On the top some research based

on self-healing research is depicted. The properties of self-healing are listed on the right.

Self-healing loop

The main design element of autonomic computing is the autonomic component[12-14]. It is kept very abstract to fit the internals of all the autonomic properties. The element comprises a manager that holds five distinct functions with individual tasks.

Monitor

The monitor gathers status information from the system through sensors and pre-processes it for the analyze task.

Analyze

This entity determines whether the received monitored information must follow a designated action. This is generally done by comparing status information to system specific thresholds.

Plan

A running system often is full of situation specific dynamics. Therefore, an accurate, sound, and planed deployment of the actions demanded by analyze is required.

Execute

Presents the entity that executes the parts of previously conceived plans on the managed element.

Knowledge

This represents the knowledge base consumed and produced by all four previously mentioned tasks.

The collaboration of the five tasks assembles the work of the manager. More precisely, the subtask of a task is to process the input and filter the output for further processing. It becomes obvious that there is a data-flow in the form of a loop among the tasks. This was called the autonomic control.

In self-healing literature, the five autonomic processes are usually reduced and included into three main stages in a loop. Kephart and Chess[14] identify them as *detection*, *diagnosis*, and *repair*. Salehie and Tahvildari[15] call it a sum of *self-diagnosing* and *self-repairing* with *discovery*, *diagnosing*, and *reacting* stages. Parashar and Hariri[16] only consider detect and recover as the stages. Figure No.4 shows the formation of the self-healing loop with the data-flow among the three stages and the environmental interfaces.

Detecting

Filters any suspicious status information received from samples and reports detected degradations to diagnosis.

Diagnosing

Includes root cause analysis and calculates an appropriate recovery plan with the help of a policy base.

Recovery

Carefully applies the planned adaptations meeting the constraints of the system capabilities and avoids any unpredictable side effects.

Self-healing states

The success of self-healing extensions depends on the distinction between system's intentional states and degraded, unacceptable states. The operating environment of self-healing extensions large-scale, unreliable systems, hold various error sources, possibly varying over time. The robustness of the self-healing alignment must not depend on a single element but the system as a whole should be able to recover from failures[17]. Thus, single element failures should have only minor impact on the whole system. In many cases there is no fine line, clearly separating acceptable from an unacceptable state. Instead, there is a momentary transmission zone in between.

The most recent model presented by Ghosh *et al*.[6], in particular, features a fuzzy transition zone with an unclear "Degraded State". This state reflects the fact that the adverse conditions of a systems cause self-healing systems to drift in a still acceptable state, however, closer to failure. This concept regards the fact that large, unpredictable systems usually do not suddenly quit operations when smaller portions fail, but continue operation with possibly considerable loss on performance. This provides recovery techniques with additional time for actions and can bring the system back on track without complete disruption. The described model is depicted in Figure No.5.

Another problem observed by Clarke and Grumberg[18] is the state explosion problem of large systems with many concurrent processes. Their observation reveals that the number of processes may cause the number of possible states to grow exponentially. The proposed solution to handle all

the possible states is to identify common properties. In the case of Alpern and Schneider[19] states are aggregated according to patterns in the execution history or in Clarke and Grumberg[18] according to equivalence classes for the running processes.

Self-healing Rules

Influenced by AI research on human behavior[20,21], Norman[22] propose a three level model based on reaction, routine, and reflection. In this model, the three levels differ in depth of processing involved between evaluation of surrounding world (affect) and interpretation of world (cognition). Later, Kephart and Walsh[23] define three different types of policies: Action, Goal and Utility Function with increasing behavioral specification that correspond the three previously presented levels. The policies related to the corresponding model level are the following:

Reaction

This is a type of policy that dictates an action to be taken on a certain incidence, similar to an IF(Condition)THEN(Action) statement. Likewise, the reaction level is defined as one where no learning occurs and immediate response is expected.

Routine

These policies define a desired state, respectively, a set of states. This implies that the system must calculate a situation depending on a set of actions to make a transition from the current to the desired state. A kin to this the routine level is defined as one, where largely routine evaluation and planning behaviors takes place.

Reflection

As a generalization of the goal policies, utility function policies connect a value to each possible state that is adjusted at runtime, depending on the current state. The reflection level is described as self-aware. It deduces the results for problem solving from information of its history, system capabilities, current system state, and current environment state.

A prototype evaluation presented by White[17] observes that goal-driven and utility function policies can be key elements to achieve a degree of self-management. Self-healing research considers recovery as a solid planned process. Simple reactive behavior might not be sufficient for the scope. Instead, to recover and also maintain the system several possible options must be balanced. The result of self-healing policies is a directly or indirectly caused set of actions moving the system towards a safe state.

Fault Categories

Fault categories and root cause analysis is a challenging task in computer networks with composite configurations. Only a category and identification of the fault allows deploying acceptable recovery strategies. Faults can affect single units or whole portions of the systems and the two types can provoke each other because of the dependencies. However, general categories of faults are available in self-healing related research. The occurrence of a fault is generally defined as an event at runtime where the current system behavior deviates from the intended. Ghosh[9] provides a comprehensive fault category for fault-tolerant systems. Coulouris[24] provide a category of faults in regard to distributed systems. Table No.1 represents a list of identified fault classes relevant to self-healing research and their possible fault resolutions.

Kopetz[25] explain about fault category in which partitions failures into dependent on value or timing by nature. A failure can be recognized consistently by all affected parties or in the worst case only inconsistently.

It becomes clear that especially in large, arbitrary systems failure detection and immediate category in most cases is not a straight forward process. A crash failure, e.g, might be classified as an omission failure because local detection is not available or affected by the failure. Thus, a detected failure might be the result of another. However, because of the many interdependencies in large systems and possible false detection and recovery strategy, self-healing technologies rely on the state model presented in Self-healing rules.

Fault Model Characteristics

The Following are typical fault model characteristics that seem relevant[26]:

Fault duration

Faults can be permanent, intermittent (a fault that appears only occasionally), or transient (due to an environmental condition that appears only

occasionally). Since it is widely believed that transient and intermittent faults outnumber permanent faults, it is important to state the fault duration assumption of a self-healing approach to understand what situations it addresses.

Fault manifestation

Intuitively, not all faults are as severe as others. Beyond that, components themselves can be designed to exhibit specific characteristics when they encounter faults that can make system-level self-healing simpler. A common approach is to design components that are fail-fast, fail-silent. However, other systems must tolerate Byzantine faults which are considered "arbitrary" faults. (It is worth noting that Byzantine faults exclude systematic software defects that occur in all nodes of a system, so the meaning of "arbitrary" is only with respect to an assumption of fault independence.) Beyond the severity of the fault manifestation, there is the severity of how it affects the system in the absence of a self-healing response. Some faults cause immediate system crashes. But many faults cause less catastrophic consequences, such as system slow-down due to excessive CPU loads, thrashing due to memory hierarchy overloads, resource leakage, file system overflow, and soon.

Fault source

Assumptions about the source of fault scan affect self-healing strategies. For example, faults can occur due to implementation defects, requirements defects, operational mistakes, and so on. Changes in operating environment can cause a previously working system to stop working, as can the onset of a malicious attack. While software is essentially deterministic, there are situations in which it can be argued that a random or "wear-out" model for failures is useful, suggesting techniques such as periodic rebooting as a self-healing mechanism. Finally, some self-healing software is designed only to withstand hardware failures such as loss of memory or CPU capacity, and not software failures.

Granularity

The granularity of a failure is the size of the component that is compromised by that fault. (The related notion of the size of a fault containment region is a key design parameter in fault tolerant computers.) A fault can cause the failure of a software module (causing an exception), a task, an entire CPU's computational set, or an entire computing site. Different self-healing mechanisms are probably appropriate depending on the granularity of the failures and hence the granularity of recovery actions.

Fault profile expectations

Beyond the source of the fault is the profile of fault occurrences that is expected. Faults considered for self-healing might be only expected faults (such as defined exceptions or historically observed faults), faults considered likely based on design analysis, or faults that are unexpected. Additionally, faults might be random and independent, might be correlated in space or time, or might even be intentional due to malicious intent.

Self-healing Systems vs. General Computing Systems

Complexity in problem determination is reducing the effectiveness of computing in many computing environments. One of the major factors contributing to the complexity in problem determination is the various ways that different parts of the system report events, conditions, errors, and alerts. For instance, examine the ways that general computers maintain logs for the system. These logs contain a variety of content in differing formats because solutions are built using disparate pieces and part, often with products from multiple vendors[27]. Figure No.6a illustrates today's general computing environments and the obstacle of combing hardware and software components in a typical solution. Most of the logging done today is focusing on reporting data that a product developer considers important for debugging the problem in a single product, as opposed to providing data to debug a solution. This inconsistency in both the format and the content that is made available by products makes it more difficult to write management tools that might ease the complexity issues. To optimize the usefulness and business value of existing and future e-business solutions, major changes and improvements in problem determination must help businesses deal with complexity, ease cross-product problem determination and automate the process of

identifying and fixing frequently occurring problems.

Autonomic computing systems are the solution to solving the problem of complexity. Refer to Figure No.6b on how autonomic computing is going to solve the problem determination of complexity. Let's begin with the resource manager. In the resource manager, various components such as applications, database, application server, servers, storage device, and networks are included. These components are a part of the troubleshooting process when a problem occurs. In Figure No.6b, each component produces multiple log files individually in its own format in various locations. Because there is no cost-effective way to change log files in legacy applications or solutions that have already been deployed, the IBM autonomic computing architecture includes adapters to translate disparate logs into the common format[27]. Adapters help keep implementation cost low, and adapters also allow business to use applications from independent software vendors that may not adopt common log formats. Log formats that are familiar enables specialist to easily look for problems in logs and take necessary actions if needed. Figure No.6b also illustrates an autonomic manager engine which automates the process that a specialist would use. For example, IBM has developed the Log and Trace Analyzer for Autonomic Computing, which enables the reading of logs in the common format, correlating the logs based on different criteria (for example, time-based or field-based (such as URLs)) and viewing the correlated log records[27]. To take the best possible action when it discovers a problem, the manager will rely upon other sources of knowledge. One such source might be a symptom service. The symptom service includes a symptom database that contains information about how to detect patterns that indicate problems, how to diagnose that a specific problem has occurred, and how to resolve that specific problem. The symptom database will include a standardized set of interfaces and data formats that facilitate the determination of actionable causes from problem data. In many instances, multiple symptom databases are possible and likely, all presented as part of a symptom service. The process of writing and populating knowledge bases, such as the symptom database, is made simpler by the existence of a common format for log data.

Symptoms will be more easily expressed using the common set of terminology and data, alleviating the need for symptoms to be coded using the nuances of how, for example, an individual product says that it has "stopped." Once the decisions are made about how to best resolve a problem, the autonomic manager may then query other managers, such as a policy engine, represented in the lower right corner of Figure No.6b, to determine which corrective actions can be taken. The policy engine matches proposed solutions against rules and policies to help ensure that an action's possible effect on business-critical processes is appropriate to the overall situation. For example, the symptom database may report that two separate actions could be used to address a symptom. The first, which would fix the problem, is to reboot a system. The second may be a temporary solution, such as increasing swapper space. In this example, if there were a policy that stated a critical system was not to be rebooted during business hours, the policy service would instruct the autonomic manager to use the temporary solution; the autonomic manager, in turn, would then provide feedback to the resource managers, which would make the necessary changes.

Related Applications to Autonomic Computing

Autonomic Computing brings new ideas and concepts in reducing complexity. There have been a number of research projects that use autonomic computing technologies in industry and academies. Some of them will be presented in this section.

Autonomic Computing Toolkit

Autonomic computing toolkit[28] presents some technologies and tools which are closely referred to the properties and general architecture of autonomic computing. This toolkit includes

Autonomic manager engine

It demonstrates the self-healing control process in the architecture of autonomic computing.

Log and Trace Analyzer

It demonstrates a partial implementation of control loops, including the part of monitoring and analyzing.

Generic log adapter

It provides a translation from log files into a common event format - Common Base Event in order for common logs to be acceptable in a autonomic computing environment.

Resource Model Builder

This Eclipse plug-in demonstrates how to build special resources into an autonomic computing environment using common resource model.

Dynamic Systems Initiative

Dynamic Systems Initiative (DSI) [29] is a Microsoft approach to reducing system complexity. As we seen in the general architecture of autonomic computing, the role of knowledge in system management is also emphasized in DSI. To benefit from the knowledge concept, DSI defines a common schema - System Definition Model - in order for other software's to be built into its operating environment. Once this model for software is created, it can be captured in system runtime, so that system is manageable autonomously.

Ocean Store

Ocean Store[30] is a global-scale persistent data storage system from the University of California at Berkeley. It uses an introspection layer to monitor and analyze network information in order to improve performance and fault management. Each data object within Ocean Store has its own GUID and is stored in distributed data location.

Other Applications

Optimal Grid[31] provides a solution of the problem of large-scale application by implementing runtime management and dynamic rebalancing. Policy Management for Autonomic Computing[32] implements an autonomic policy management. The Adaptive Enterprise[33] provides an enterprise infrastructure used to manage enterprise knowledge in real time.

Challenges of Autonomic Computing

During the implementation of autonomic computing some related practices[29,33] show that self-managing, adaptive computing systems can be realized, and have a great perspective. However, developing those autonomic systems are "beyond the boundaries of traditional computer sciences"[34] and requires a global cooperation of research in diverse fields. The architecture of autonomic computing simplified this work in a large scale, but caused also some new challenges. These challenges can be divided into three categories: standardization challenges, algorithms and methods challenges and management challenges.

Standardization Challenges

Autonomic computing is an open computing; it needs a common, standard model in multidimensional.

Representation of autonomic element needs standardization. An autonomic element may represent a special business or scientific objective, and its services should be shared by other autonomic elements. Thus an open, standard model for autonomic element is needed to design autonomic elements.

Knowledge management needs standardization. In the architecture of autonomic computing knowledge is shared in the implementation of managed loop. In the analyze phase, autonomic computing needs to understand the meaning of monitored data autonomously and selects the useful information from them. This requires (a) a common log format for the understanding of monitored data; and (b) a common event correlation to determine useful expressions.

Services sharing and parameters' negotiation between different autonomic elements need standardization. Different autonomic elements should operate in an unpredictable environment as a whole. They need to utilize their resources efficiently and to be aware of presence of other autonomic elements and external environment. To achieve it, services should be discovered autonomously and be shared within those autonomic elements. This requires a standardization of negotiation protocol, for example, service discovery protocol and service utilization protocol.

System wide collaboration needs standardization. Various autonomic elements collaborate with each other and form a great autonomic computing system. The coordination between different autonomic elements is usually policy-based. These policies should (a) exactly express the goal of the complex system; and (b) be understandable by underlying

autonomic elements. Some projects[35] attempt to solve this problem, but a standardization of policy in autonomic computing is still required.

Algorithms and Methods Challenges

Autonomic computing needs a global co-operation in diverse fields. To develop autonomic computing, some algorithms and methods should be newly researched.

Learning algorithm

Learning algorithm is closely tied to autonomic computing. From problem determination and autonomic remediation to system wide optimization, learning algorithms are used everywhere, but under new conditions, namely, critical services should not be disrupted. The exploration of learning algorithms is different from the traditional one. How exact an error can be allowed, how to improve the performance of learning process, and how to coordinate different learning process, all of that remain a research challenge.

Process co-ordination methods

Autonomic computing system consists of a large scale of autonomic elements. Each of them represents a different objective (i.e, database, webserver) and expresses different optimization criteria. Within an autonomic element there run also many processes. How to coordinate such large number of processes to optimize, configure and reconfigure remains a research challenge.

Attack detection methods

With autonomic computing exchange of information is accomplished in a autonomous way. Autonomic element need not only to understand the incoming information but also to detect active attacks and protect itself against those attacks.

Management Challenges

The goal of autonomic computing is to reduce the tasks of nowadays administrators. To achieve it, there need new techniques to monitor and visualize what autonomic computing and its autonomic elements do. These techniques must be "sufficiently expressive of preferences regarding cost vs. performance, security, risk and reliability"[34].

Table No.1: Fault Categories

S.No	Fault Classes	Possible Failure Resolution
	Crash failure	State recovery and restart
1	Fail-stop	Stable storage status reconstruction and partition of remaining work
2	Omission	Re-route, retransmission
3	Transient	Recovery of side effects
4	Timing and Performance	Re-assignment of task
5	Security	Behavior dependent
6	Arbitrary	Reconstruction, resend and ignore

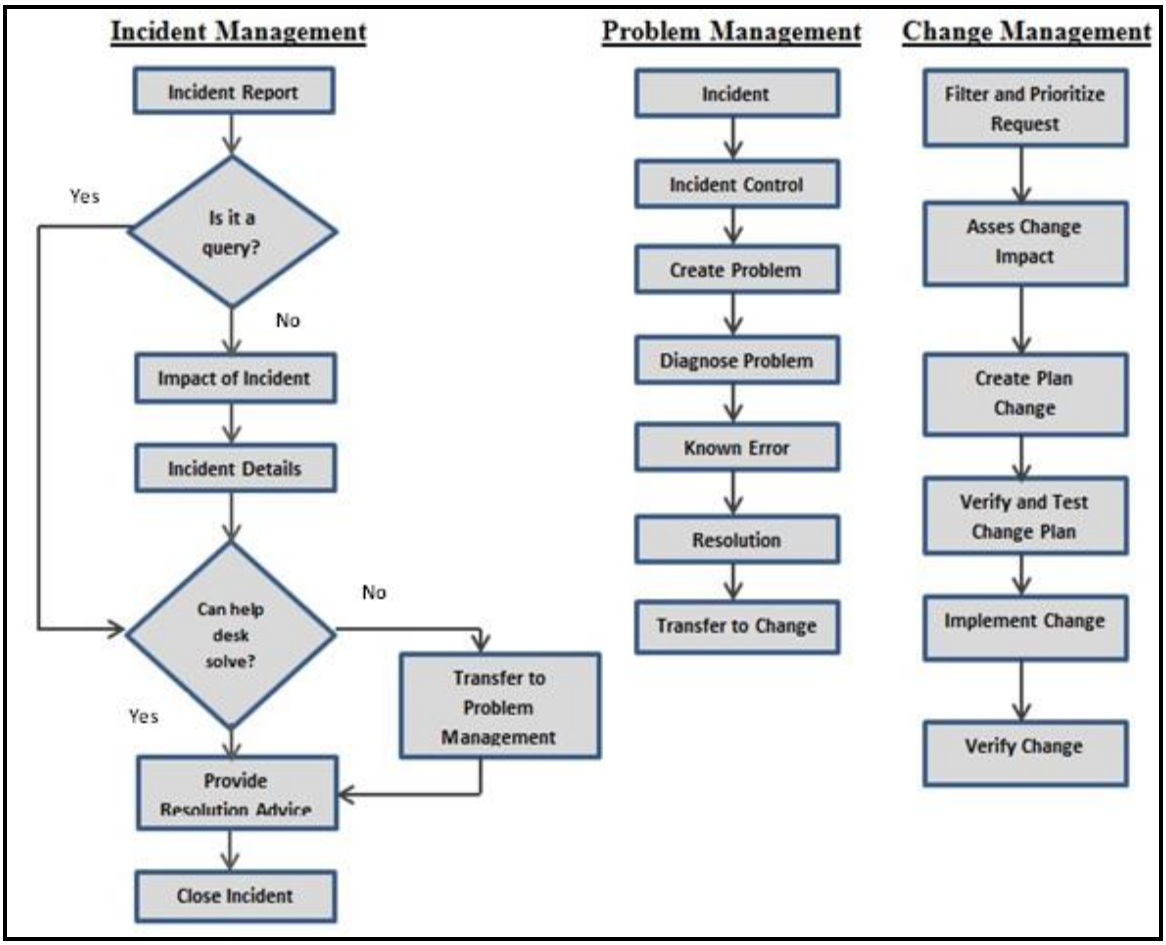

Figure No.1: Typical procedures implemented in various IT organizations

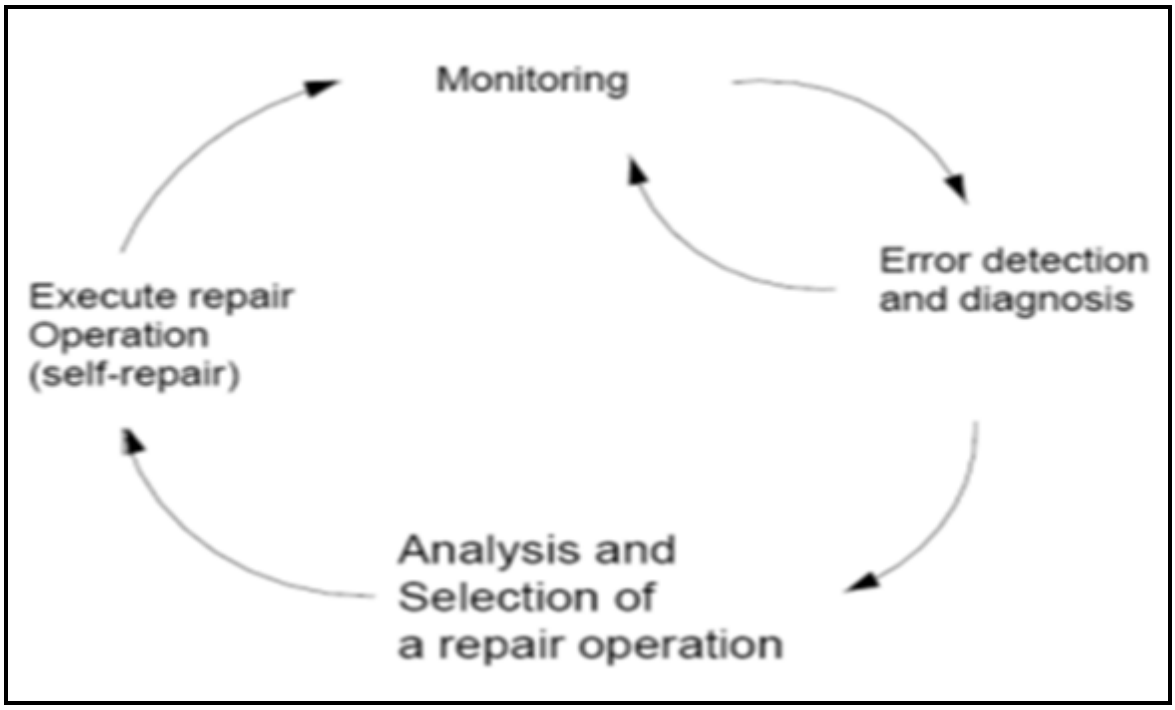

Figure No.2: Self-healing System Process

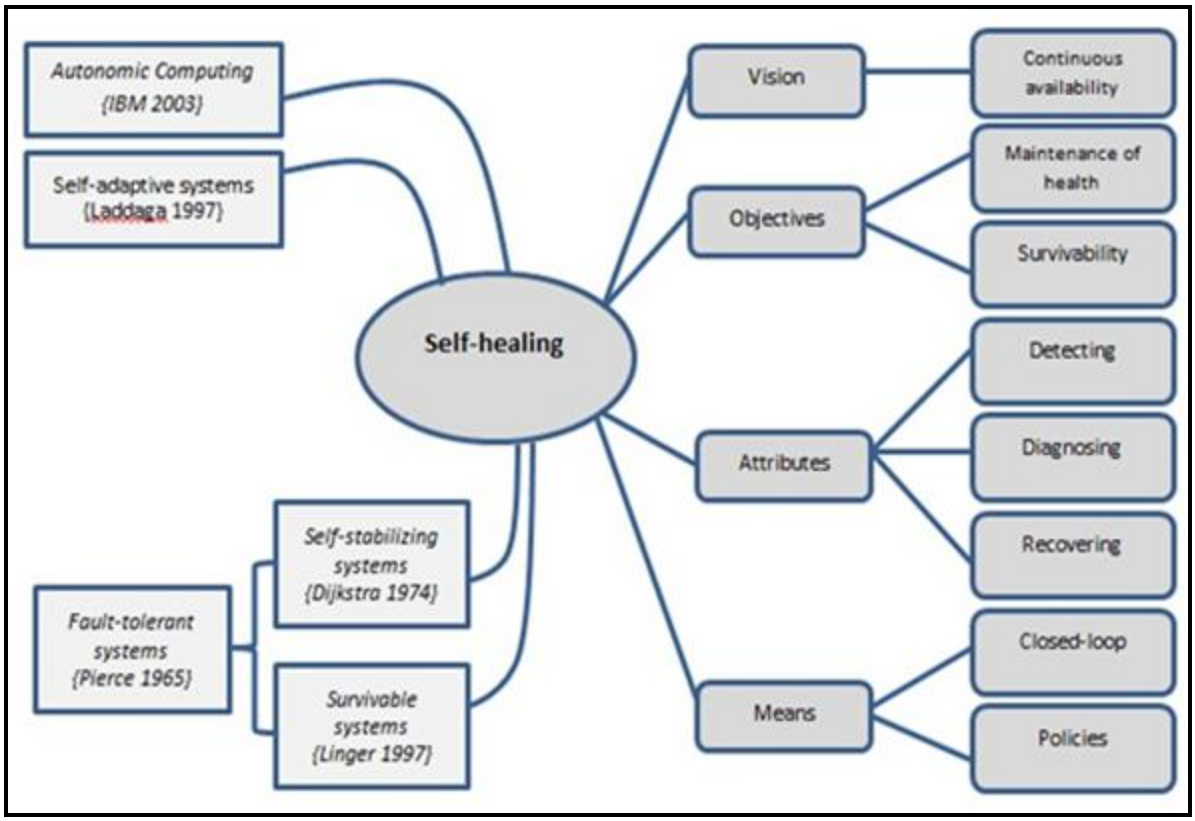

Figure No.3: Relations and properties of self-healing research

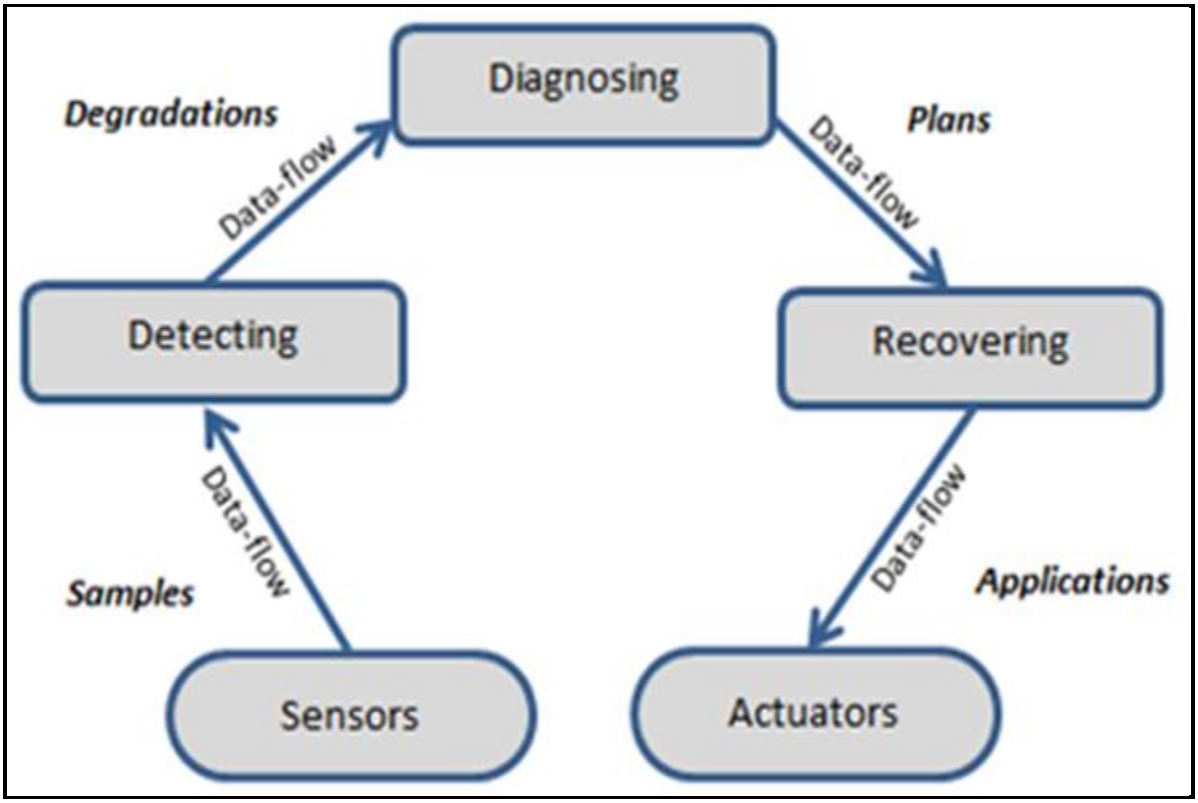

Figure No.4: Staged loop of self-healing

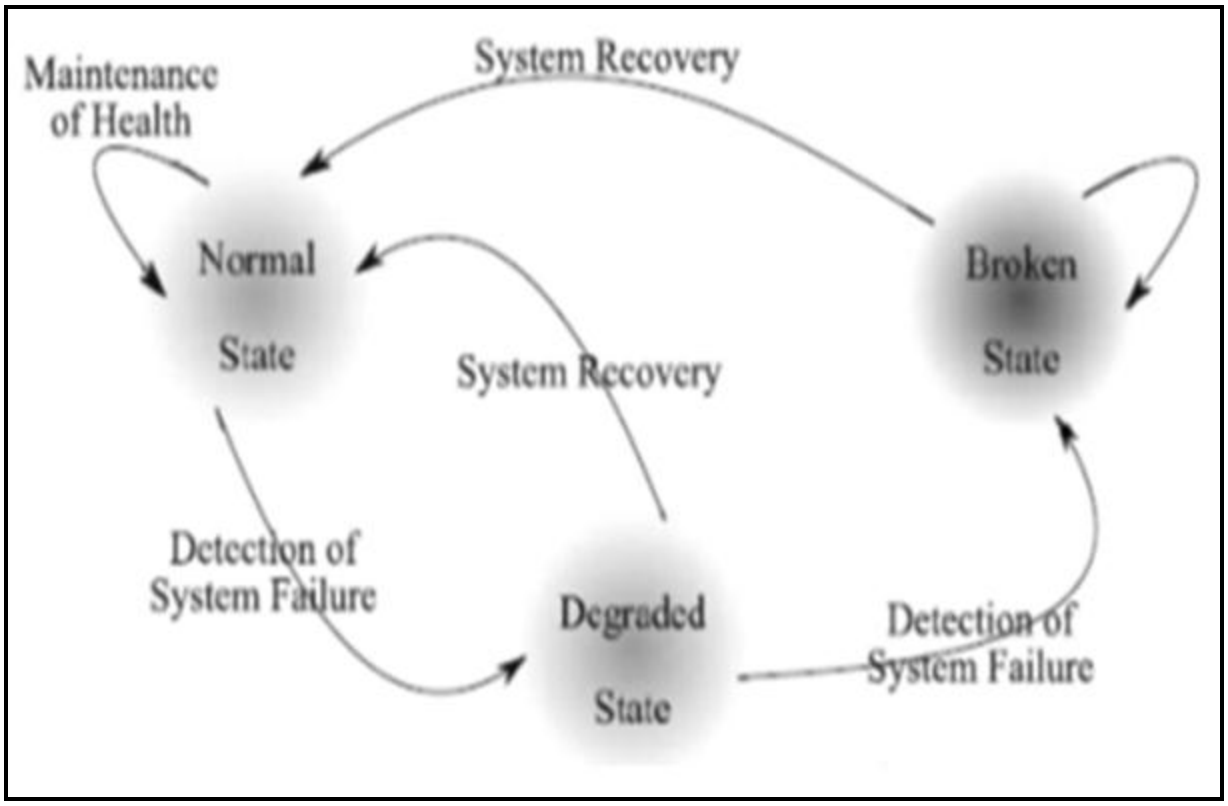

Figure No.5: State diagram of self-healing

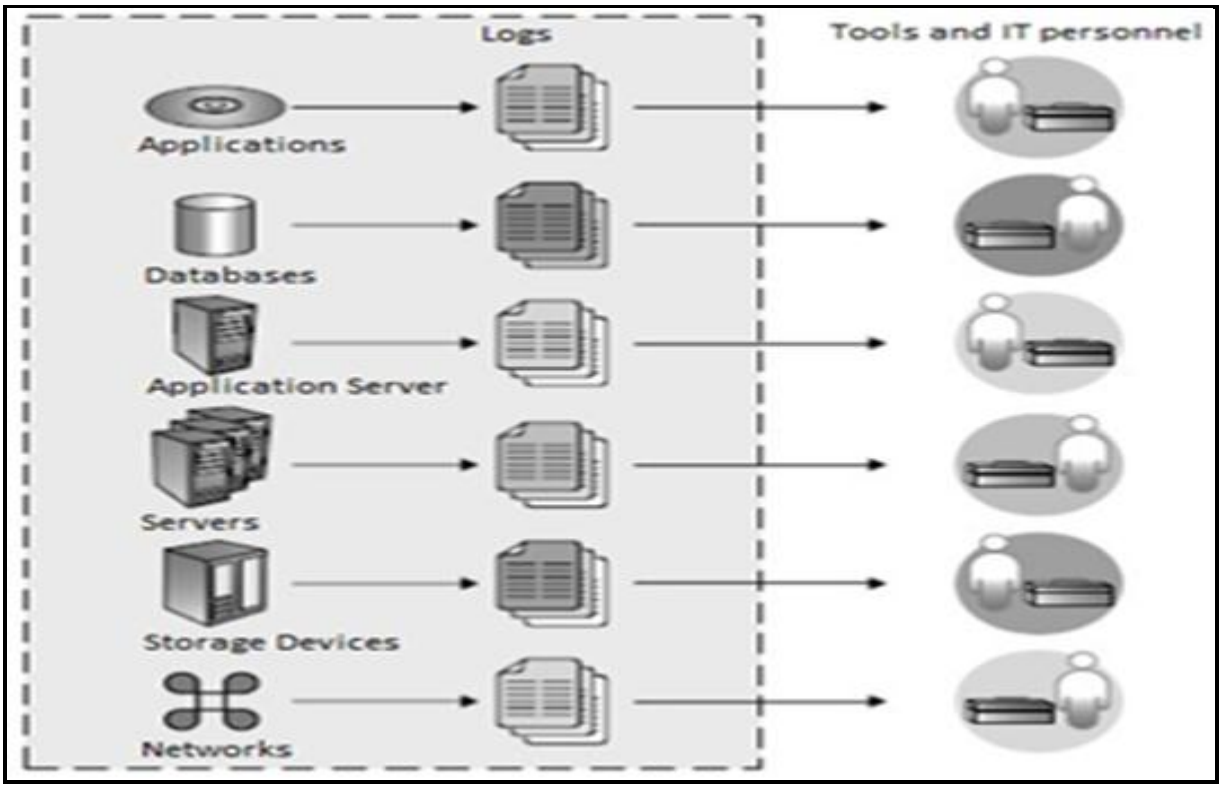

Figure No.6a: General Computing System

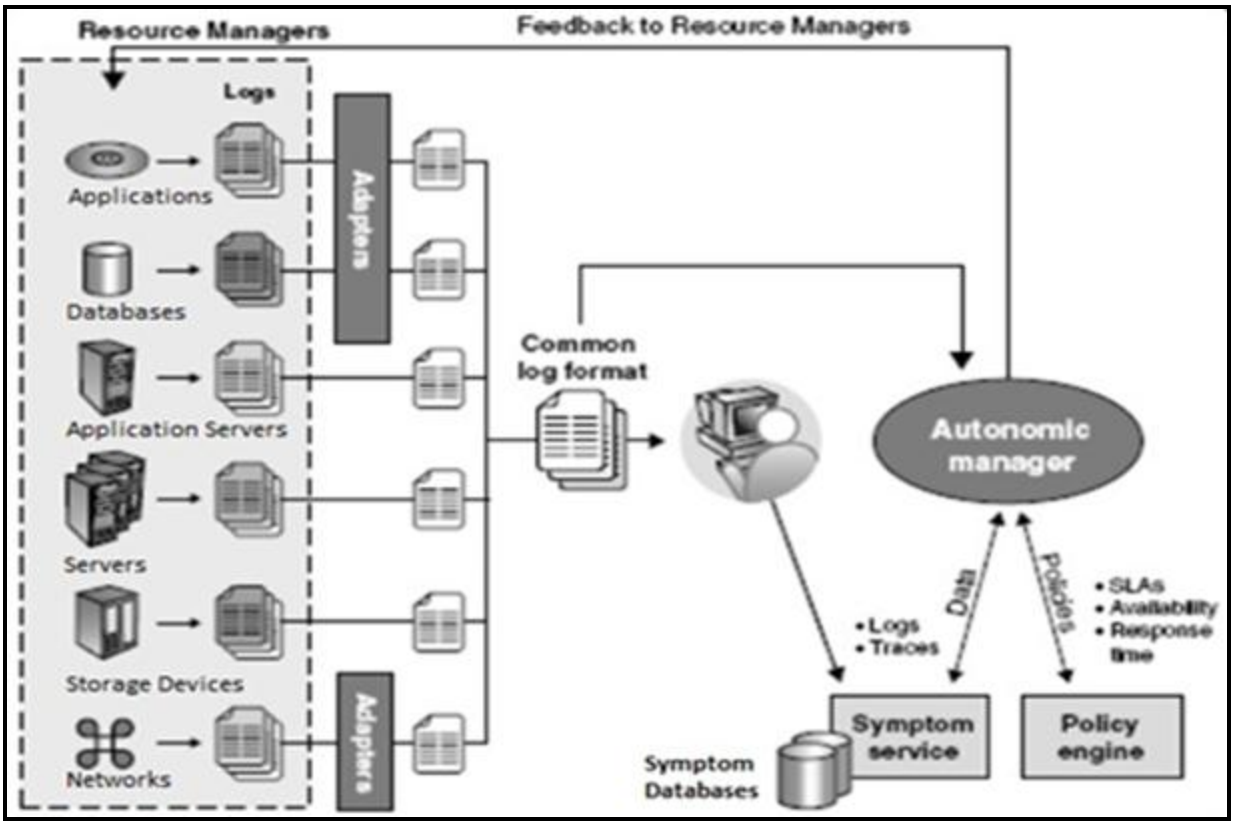

Figure No.6b: Autonomic Computing Self-Healing System

CONCLUSION

Autonomic computers that self-heal themselves basically endure a process in order to maintain satisfactory quality of service of the principle system during runtime in the presences of any fault. The process is a closed loop cycle that consists of a monitoring cycle, an error detection and diagnosis cycle, analysis and a selection of a repair operation cycle, and an execute repair operation cycle. Each self-healing system process has a fault model in terms of what faults they are expected to be able to self-heal because without a fault model there is no way to assess whether a system actually can heal itself in situation of interest. There are many advantages of autonomic computing systems. As well as addressing complexity, autonomic computing offers the promise of a lower cost of ownership and a reduced maintenance burden as systems become self-managing. Nonetheless, there are some limitations also to this vision. The challenges of re-engineering today's systems of systems away from the complexity dilemma toward tomorrow's persuasive and ubiquitous computations and communications will require unifying standards, new economic models and trust of the users, as well as innovations to address the hard technical issues. IBM's vision of autonomic computing is much like a journey than a destination. The journey to achieve the ultimate vision of autonomic systems has just begun. This new era of computing is greater than any single IT company. Many universities are exploring various aspects of autonomic computing such as self-configuring, self-healing, self-optimizing, self-protecting, grid computing, and routing. Increasing constraints on resources and greater focus on the cost of operations, has led NASA and others to utilize autonomy. Achieving overall autonomic behaviors remains an open and significant challenge, which will be accomplished through a combination of process changes, skills evolution, new technologies and architecture, and open industry standards.

ACKNOWLEDGEMENT

I'm very thankful to Eritrea Institute of Science and Technology, Asmara, Eritrea, North East Africa and I would also like to thank the Management of International Computing Council, for provided the necessary facilities to carry out this Research work.

CONFLICT OF INTEREST

We declare that we have no conflict of interest.

BIBLIOGRAPHY

1. Sterritt Roy M, Parashar H Tianfiels and Unland R. A concise introduction to autonomic computing, *Advanced Engineering Informatics,* 2(8), 2005, 320-567.
2. Ganek A G and Corbi T A. The dawning of the autonomic computing era, *IBM System Journal,* 42(1), 2003, 245-589.
3. Tosi and Davide. "Research perspective in self-healing systems", *IEEE Computing,* 4(1), 2004, 10-45.
4. Ganek A G, Corbi T A. The dawning of the autonomic computing era, *IBM Syst J,* 42(1), 2003, 5-18.
5. Pierce W. Failure-tolerant computer design, *Academic Press, New York,* 3(4), 1965, 120-340.
6. Ghosh D, Sharman R, Raghav Rao H, Upadhyaya S. Self-healing systems survey and synthesis, *Decis Support Syst,* 42(4), 2007, 2164-2185.
7. Ellison R, Fisher D, Linger R, Lipson H, Longstaff T, Mead N. Survivability: protecting your critical systems, *Internet Comput IEEE,* 3(6), 1999, 55-63.
8. Linger R, Mead N, Lipson H. Requirements definition for survivable network systems, *ICRE'98*, 2(5), 1998, 6-10.
9. Merideth M. Enhancing survivability with proactive fault-containment, *DSN Student Forum, Citeseer*, 20(3), 2003, 300-670.
10. Akoglu A, Sreerama reddy A, Josiah J. FPGA based distributed self-healing architecture for reusable systems, *Cluster Comput,* 12(3), 2009, 269-284.
11. Corsava S, Getov V. Intelligent architecture for automatic resource allocation in computer clusters, *IEEE Computer Society, Washington, DC*, 17(3), 2003, 201-1.
12. Huebscher M C, McCann J A. A survey of autonomic computing, *Degrees, models, and applications*, 8(2), 2008, 120-300.
13. IBM: An architectural blueprint for autonomic computing, *IBM*, 15(8), 2005, 456-536.
14. Kephart J O, Chess D M. The vision of autonomic computing, *Comput IEEE Comput Soc Press,* 36(1), 2003, 41-50.
15. Salehie M, Tahvildari L. Self-adaptive software: landscape and research challenges, *ACM Trans Auton Adapt Syst,* 4(2), 2009, 1-42.
16. Parashar M, Hariri S. Autonomic computing: an overview. In: Unconventional programming paradigms, *Springer, Berlin*, 14(5), 2005, 247-259.
17. White S, Hanson J, Whalley I, Chess D, Kephart J: An architectural approach to autonomic computing, *In: Proceedings international conference on autonomic computing*, 2(3), 2004, 2-9.
18. Clarke E M, Grumberg O. Avoiding the state explosion problem in temporal logic model checking, ACM Symposium on Principles of distributed computing, *ACM, New York*, 6(1), 1987, 294-303.
19. Alpern B, Schneider F B. Verifying temporal properties without temporal logic, *ACM Trans Program Lang Syst*, 11(1), 1989, 147-167.
20. Picard R W. Affective computing, *The MIT Press, Cambridge*, 23(4), 1997, 570-780.
21. Sloman A, Croucher M. Why robots will have emotions, *In: Proceedings IJCAI*, 4(3), 1981, 456-567.
22. Norman D A, Ortony A, Russell D M. Affect and machine design: lessons for the development of autonomous machines, *IBM Syst J*, 42(1), 2003, 38-44.
23. Kephart J, Walsh W. An artificial intelligence perspective on autonomic

computing policies, *POLICY*, 5(1), 2004, 3-12.
24. Coulouris G, Dollimore J, Kindberg T. Distributed systems: concepts and design, *Addison- Wesley Longman Publishing Co., Inc., Boston*, 3(2), 1994, 210-250.
25. Kopetz H. Real-time systems: design principles for distributed embedded applications, *Springer, Berlin*, 20(4), 1997, 234-467.
26. Philip Koopman. Elements of the Self-Healing System Problem Space, *ICSE-WADS*, 4(3), 2003, 34-123.
27. IBM Corporation: Automating problem determination: a first step towards self-healing computer systems, *IBM*, 2003.
28. IBM: Autonomic computing toolkit, (available via http://www-128.ibm.com/developerworks/autonomic/sublink5.html).
29. Microsoft: System definition model overview white paper, (available via http://www.microsoft.com/windowsserversystem/dsi/sdmwp.mspx).
30. Berkeley: Oceanstore, (available via http://oceanstore.cs.berkeley.edu).
31. IBM: Optimalgrid, (available via http://www.alphaworks.ibm.com/tech/optimalgrid).
32. IBM: Policy management for autonomic computing, (available via http://www.alphaworks.ibm.com/tech/pmac).
33. HP: Adaptive enterprise strategy, (available via http://h41111.www4.hp.com/enterprise/de/de/ae/index.html).
34. Kephart J O. Research challenges of autonomic computing, (available via http://portal.acm.org/citation.cfm?id=1062464&dl=&coll=GUIDE&CFID=15151515&CFTOKEN=6184618).

Please cite this article in press as: Raja Adeel Ahmed *et al.* Methodologies of Self-Healing and System Confronts (The Autonomic Computing), *International Journal of Engineering and Robot Technology*, 1(2), 2014, 41 - 55.