A Survey of Autonomic Computing Systems

Mohammad Reza Nami, Koen Bertels
Computer Engineering Laboratory, Delft University of Technology

Abstract— **The evolution of networks and Internet has introduced highly scalable and available services making operational environments more complex. The increasing complexity, cost and heterogeneity of distributed computing systems have motivated researchers to investigate new ideas to cope with the management of this complexity. One such idea is *autonomic computing*. This paper provides a thorough survey of autonomic computing systems, presenting their characteristics, effects on quality factors, their building block architecture and challenges.**

I. Introduction

Data and programs in centralized applications are kept at one site and this is conceived as a bottleneck in performance and availability of remote information in desktop computers. Distributed systems were emerged to remove this flaw. During 1990s, distributed databases and client-server packages were used for information exchange between remote desktop computers. In these years, Distributed Computing Systems (DCSs) consisted of different computers connected to each other and located at geographically remote sites. This was the starting point for emerging concepts such as Peer, Peer-to-Peer (P2P) Computing [9], Agents [15], and Grid Computing [11]. The evolution of networks and Internet presented highly scalable and available services which has made environments more complex. This complexity has increased the cost and errors of managing IT infrastructures. The skilled persons who manage these systems are expensive and cannot manage them in configuration, healing, optimization, protection and maintenance. Moreover, IT managers look for ways to improve the Return On Investment (ROI) by reducing the Total Cost of Ownership (TCO), improving Quality of Services (QoSs) and reducing the cost for managing of IT complexity. A study shows that 25 to 50 percent of IT resources are spent on problem determination and almost half of the total budget is spent to prevent and recover systems from crashes [12].

All these issues have motivated researchers to investigate a new idea to cope with the management of complexity in IT industry and self-management systems have been introduced. On March 8, 2001, Paul Horn presented importance of these systems by introducing Autonomic Computing Systems (ACSs) to the National Academy of Engineering at Harvard University. Some benefits of autonomic computing include reduction of costs and errors, improvement of services and reduction of complexity. We are going to survey these issued in more depth in this paper.

The rest of paper is organized as follows. Related works are surveyed in section 2. In section 3, we present an overview of ACSs including definitions, benefits, and their characteristics. Section 4 describes Autonomic Elements (AEs) architecture as the building blocks in ACSs. In section 5, some challenges such as robustness, learning, and relationships among AEs are discussed. Finally, we present conclusions and further researches.

II. Related Work

On March 8, 2001, Paul Horn presented a link between pervasiveness and self-regulation in body 's autonomic nervous system and introduced ACSs to the National Academy of Engineering at Harvard University. With choosing the term *autonomic*, researchers attempted to make autonomic capabilities in computer systems with the aim of decreasing the cost of developing and managing them. Many researchers have studied this subject since 2001. Their studies have been categorized as follows:

- **Architecture and environment for ACSs:** S. White in [17], and R. Sterritt and D. Bustard in [13] have described some general architectures for ACSs and their necessary elements called autonomic elements.
- **Studying criteria for evaluating ACSs:** J. A. McCann and M. C. Huebscher in [8] have proposed some metrics to evaluate ACSs like cost and adaptability. Some performance factors such as security and availability have been discussed by others [1].
- **ACS properties:** These are self-optimization [12], self-configuration [15], self-healing [4], and self-protection [13]. Of course, the IBM Group in [3] has stated a general schema for ACSs and their characteristics.
- **Evaluation ACS from software engineering vision:** P. Leaney, A. MacArthur, and J. Leaney [7] have established the role of autonomic computing in developing software projects.
- **Challenges in ACSs:** Many researches [5] have been done in this context.
- **AC Products:** Different projects and products have been developed in both by the industry and the academic. M. Salehie and L. Tahvildari have outlined some of these products in [12].

From another view, researches carried out in this field can be categorized in two groups as the follows:

- **Group 1:** Researches which describe technologies related to autonomic computing.
- **Group 2:** Researches which attempt to develop autonomic computing as an unified project.

However, the lake of appropriate tools for managing the complexities in large scale distributed systems has encouraged researchers to designing and implementing ACSs features.

III. Overview

In this section, initial concepts about definitions and characteristics are discussed.

A. *Definition and characteristics*

The autonomic concept is inspired by the human body 's autonomic nervous system. The human body has good mechanisms for repairing physical damages. It is able to effectively monitor, control, and regulate the human body without external intervention. An autonomic system provides these facilities for a large-scale complex heterogeneous system. An ACS is a system that manages itself. According to Paul Horn 's definition, an ACS is a self-management system with eight elements. Self-configuration means that An ACS must dynamically configure and reconfigure itself under changing the conditions. Self-healing means that An ACS must detect failed components, eliminate it, or replace it with another component without disrupting the system. On the other hand, it must predict problems and prevent failures. Self-optimization is the capability of maximizing resource allocation and utilization for satisfying user requests. Resource utilization and work load management are two significant issues in self-optimization. An ACS must identify and detect attacks and cover all aspects of system security at different levels such as the platform, operating system, applications, etc. It must also predict problems based on sensor reports and attempt to avoid them. It is called as Self-protection. An ACS needs to know itself. It must be aware of its components, current status, and available resources. It must also know which resources can be borrowed or lended by it and which resources can be shared. It is Self-awareness or Self-knowledge property. An ACS must be also aware of the execution environment to react to environmental changes such as new policies. It is called as context-awareness or environment-awareness. Openness means that An ACS must operate in a heterogeneous environment and must be portable across multiple platforms. Finally, An ACS can anticipate its optimal required resources while hiding its complexity from the end user view and attempts to satisfy user requests. Self-configuration, self-healing, self-optimization, and self-protection are considered as major characteristics and the rest as minor characteristics. As mentioned above, the aim of AC is to improve the system abilities. Therefore, AC characteristics affect various measurements of quality such as usability, functionality, reliability, maintainability, and portability.

B. *A survey of different definitions*

The aim of this survey is to identify all the possible definitions about ACSs. The common professional researchers in this field have considered for this survey. They are first author in their publications. The following definitions for autonomic computing are presented:

- Kephart [5]: Major characteristics and self-managing.
- Chess [1]: Major characteristics.
- Tivoli IBM [3]: Major and minor characteristics.
- Sterritt [14]: Major characteristics, self-governing, self-adapting, self-managing, self-recovery, and self-diagnosis of faults.
- Tianfield [16]: Self-mechanism including major characteristics, self-planning, self-learning, self-scheduling, self-evolution, and etc.
- Parashar [11]: Major characteristics, self-adapting.
- Murch [10]: Major and minor characteristics.
- Tesauro [15]: Goal-driven self-assembly, self-healing, and real-time self-optimizing.
- De Wolf [18]: Major characteristics.
- White [17]: Major characteristics and self-managing.
- Ganek [2]: Major and minor characteristics.

with closer examination of the papers, it is found that these definitions are derived from the eight elements proposed by Horn in 2001. For example, D. M. Chess et al have used the term 'self-configuration' similar to Horn 's definition and have presented 'self-assembly' property in Unity as an autonomic computing product. Some terms such as self-tuning [16] and self-adapting [11] are conceptually similar to existing characteristics. 'Self-managing' in Kephart [6], White et al [17], and Sterritt [14] has been used as a popular property and major characteristics have been its subset. A trend which emerges from the analysis of the definitions is that some researchers have defined the same concept with different terms. For example, Tesauro et al [15] have defined self-assembly as a concept similar to self-configuration for an autonomic computing product. 'Environment-awareness' is used by Sterritt [13] to describe the sixth element of Horn 's definition, but more researchers have used the term 'context-awareness' to describe the same concept. As described above, all autonomic computing characteristics have been mentioned in almost half of the papers surveyed. While major characteristics have been used to describe an autonomic computing system in every paper, anticipatory has been represented in few papers surveyed.

IV. Toward Autonomic Element Architecture

The goal of an autonomic computing architecture is to reduce intervention and carry out administrative functions according to predefined policies. Moving from manual to autonomic systems is introduced in a step-by-step manner by Tivoli group in IBM. ACSs also can make decisions and manage themselves in three scopes: resource element scope, group of resource elements scope, and business scope. In resource element scope, individual components such as servers and databases manage themselves. In group of resource elements scope, pools of grouped resources that work together perform self-management. For example, a pool of servers can adjust work load to achieve high performance. Finally, overall business context can be self-managing. It is clear that increasing the maturity levels of AC will affect on level of making decision. The path to AC consists of five levels: basic, managed, predictive, adaptive, and autonomic. They are explained in the following [[10]]:

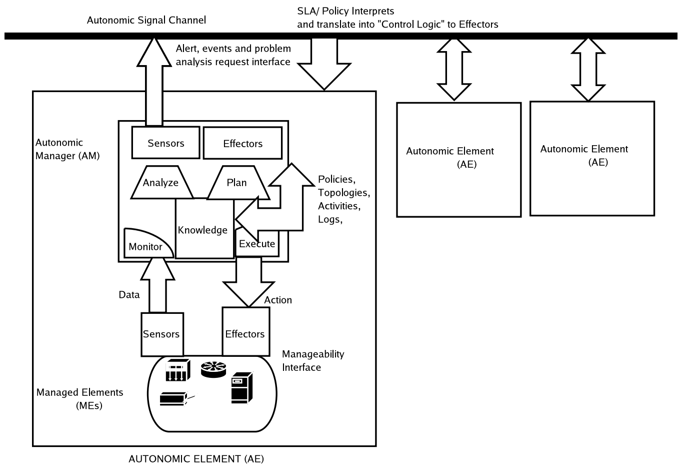

Fig. 1. Autonomic Element architecture

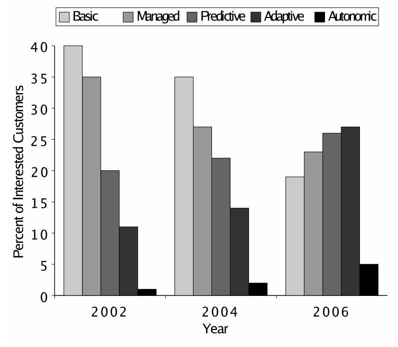

Fig. 2. Estimate of people trends toward autonomic products

- **Basic Level:** At this level, each system element is managed by IT professionals. Configuring, optimizing, healing, and protecting IT components are performed manually.
- **Managed Level:** At this level, system management technologies can be used to collect information from different systems. It helps administrators to collect and analyze information. Most analysis is done by IT professionals. This is the starting point of automation of IT tasks.
- **Predictive Level:** At this level, individual components monitor themselves, analyze changes, and offer advices. Therefore, dependency on persons is reduced and decision making is improved.
- **Adaptive Level:** At this level, IT components can individually or group wise monitor, analyze operations, and offer advices with minimal human intervention.
- **Autonomic Level:** At this level, system operations are managed by business policies established by the administrator. In fact, business policy drives overall IT management, while at adaptive level; there is an interaction between human and system.

Autonomic Elements (AEs) are the basic building blocks of autonomic systems and their interactions produce self-managing behavior. We can consider AEs as software agents and ACSs as multi-agent systems. Each AE has two parts: Managed Element (ME) and Autonomic Manager (AM). In fact, ACSs are established from Managed Elements (MEs) whose behaviors are controlled by Autonomic Managers (AMs). AMs execute according to the administrator policies and implement self-management. An ME is a component from system. It can be hardware, application software, or an entire system. Sensors retrieve information about the current state of the ME and then compare it with expectations that are held in knowledge base by the AE. The required action is executed by effectors. Therefore, sensors and effectors are linked together and create a control loop. Autonomic Managers (AMs) are the second part of an AE. An AM uses a manageability interface to monitor and control the ME. It has four parts: monitor, analyze, plan, and execute. The monitor part provides mechanisms to collect information from a ME, monitor it, and manage it. Monitored data is analyzed. It helps the AM to predict future states. Plan uses policy information and what is analyzed to achieve goals. Policies can be a set of administrator ideas and are stored as knowledge to guide AM. Plan assigns tasks and resources based on the policies, adds, modifies, and deletes the policies [17]. AMs can change resource allocation to optimize performance according to the policies. Finally, the execute part controls the execution of a plan and dispatches recommended actions into ME. These four parts provide control loop functionality. Communications between AMs provide self-managing and context-awareness. External behavior of AEs is related to relationships among them. Figure 1 shows detailed architecture of AEs in an AC environment. AMs can be linked together via an autonomic signal channel. The Tivoli group has also presented an estimation of people tending towards the adoption of autonomic operations from 2002 to 2006. Figure 2 shows results of this estimate.

V. Autonomic Computing Challenges

Since autonomic computing is a new concept in large-scale heterogeneous systems, there are different challenges and issues. Some of them have been explained in the following:

A. Issues in Relationships among AEs

Relationships among AEs have a key role in implementing self-management. These relationships have a life cycle consisting of specification, location, negotiation, provision, operation, and termination stages. Each stage has its own challenges [6]. Expressing the set of output services that an AE can perform and the set of input services that it requires in a standard form, as well as establishing the syntax and semantics of standard services for AEs, can be a challenge in specification. As an AE must dynamically locate input services that it needs and other elements that need its output services must dynamically locate this element with looking it up, AE reliability can be a research area in location stage. AEs also need protocols and strategies to establish rules of negotiation and to manage the flow of messages among the negotiators. One of challenges is for the designer to develop and analyze negotiation algorithms and protocols, then determine which negotiation algorithm can be effective. Autonomated provision can also be a research area for next stage. After agreement, the AMs of both AEs control the operation. If the agreement is violated, different solutions can be introduced. This can be a research area. Finally, after both AEs agree to terminate the negotiated agreement, the procedure should be clarified.

B. Learning and Optimization Theory

How can we transfer the management system knowledge from human experts to ACSs? The master idea is that by observing that how several human experts solve a problem on different systems and by using traces of their activities, a robust learning procedure can be created. This procedure can automatically perform the same task on a new system. Of course, facilitating the knowledge acquisition from the human experts and producing systems that include this knowledge can be a challenge. One of the reasons for the success of ACSs is their ability to manage themselves and react to changes. In short, in sophisticated autonomic systems, individual components that interact with each other, must adapt in a dynamic environment and learn to solve problems based on their past experiences. Optimization can be a challenge too, because in such systems, adaptation changes behavior of agents to reach optimization. The optimization is examined at AE level.

C. Robustness

There are many meanings for robustness. Robustness has been served in various sciences and systems such as ecology, engineering, and social systems. We can interpret it as stability, reliability, survivability, and fault-tolerance, although it does not mean all of these. Robustness is the ability of a system to maintain its functions in an active state, and persist when changes occur in internal structure of the system or external environment. Some often mistake it with stability. Although both stability and robustness focus on persistence, robustness is broader than stability. It is possible that components of a system are not themselves robust, but interconnections among them make robustness at the system level. A robust system can perform multiple functionalities for resistance, without change in the structure. With the design of instructions that permit systems to preserve their identity even when they are disrupted, the robustness in systems can be increased. Robustness is one of grand scientific challenges which can be also examined in programming.

VI. Conclusions and Future works

In a distributed computing system, users and multiple computers are interconnected in an open, transparent, and geographical large-scale system. Therefore, development and management of these systems are master problems for IT professionals. IBM proposed Autonomic Computing Systems (ACSs) as a solution. ACSs manage themselves. Four major characteristics of such systems include self-configuration, self-optimization, self-protection, and self-healing. To achieve them, ACSs have four minor characteristics as self-awareness, context-awareness, openness, and anticipatory. Autonomic Elements (AEs) provide self-managing behavior in ACSs. They are the building blocks of ACSs and their interactions produce self-managing behavior. The various parts of AEs have been automated with evolution of AC levels. The engineering and scientific challenges raised in this field include robustness, learning, and relationships among AEs.

In this paper, a survey of autonomic computing systems and their importance was presented. As future researches, the following topics can be proposed in autonomic distributed computing domain:

1) Performance evaluation of applying the autonomic behavior in a distributed computing system model.
2) Designing an autonomic manager in multi-layer P2P form, so that autonomic behavior and management information as a knowledge base are stored in separated layers.
3) Studying languages which develop autonomic management behavior in a distributed computing environment.
4) Implementing a self-healing system in a virtual organization wherein some partners may fail.

References

[1] D. M. Chess, C. Palmer, and S. R. White. Security in an autonomic computing environment. In *IBM System Journal*, volume 42, pages 107–118, January 2003.

[2] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. In *IBM System Journal*, volume 42, pages 5–18, January 2003.

[3] IBM Corporation Software Group. The Tivoli software implementation of autonomic computing guidelines. In *Available at http://www-03.ibm.com/autonomic/pdfs/br-autonomic-guide.pdf*, 2002.

[4] S. Hariri and M. Parashar. Autonomic Computing: An overview. In *Springer-Verlag Berlin Heidelberg*, pages 247–259, July 2005.

[5] J. O. Kephart. Research challenges of autonomic computing. In *Proceedings of the 27th International Conference on Software Engineering*, pages 15–22, May 2005.

[6] J. O. Kephart and D. M. Chess. The vision of autonomic computing. In *IEEE Computer*, volume 36, pages 41–50, January 2003.

[7] P. Leaney, A. MacArthur, and J. Leaney. Defining Autonomic computing: A software engineering perspective. In *Australian Software Engineering Conference (ASWEC'05)*, 2005.

[8] J. A. McCann and M. C. Huebscher. Evaluation issues in autonomic computing. In *Proceedings of Grid and Cooperative Computing workshop(GCC)*, volume 15, pages 597–608, October 2004.

[9] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. In *Proceedings of the Second International Conference on Peer-to-Peer Computing*, pages 1–51, July 2002.

[10] R. Murch. Autonomic Computing. In *Prentice-Hall*, pages 0–20:25–40, October 2004.

[11] M. Parashar, Z. Li, H. Liu, V. Matossian, and C. Schmidt. Enabling Autonomic Grid Applications: Requirements, Models and Infrastructures. In *Self-Star Properties in Complex Information Systems, Lecture Notes in Computer Science, Springer Verlag*, volume 3460, 2005.

[12] M. Salehie and L. Tahvildari. Autonomic Computing: emerging trends and open problems. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–7, July 2005.

[13] R. Sterritt and D. Bustard. Towards an autonomic computing environment. In *14th International Workshop on Database and Expert Systems Applications (DEXA '03)*, pages 694–698, September 2003.

[14] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland. A concise introduction to autonomic computing. In *Advanced Engineering Informatics*, volume 19, pages 181–187, January 2005.

[15] G. Tesauro and et al. A Multi-agent systems approach to autonomic computing. In *IBM Press*, pages 464–471, March 2004.

[16] H. Tianfield. Multi-agent autonomic architecture and its application in e-medicine. In *IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003)*, pages 601–604, October 2003.

[17] S. White and et al. An architectural approach to autonomic computing. In *Proceedings International Conference on Autonomic Computing (ICAC'04), NewYork, USA*, pages 2–9, May 2004.

[18] T. De Wolf and T. Holvoet. Evaluation and comparison of decentralised autonomic computing systems. In *Department of Computer Science, K.U.Leuven, Report CW 437, Leuven, Belgium*, March 2006.