Autonomic Computing: a Study.

Stefan Contiu

stefan.contiu@gmail.com

Technical University of Cluj-Napoca - 2014

This study presents the most important characteristics of Autonomic Computing, with emphasis on the Self-Management vision for increasingly complex large distributed systems. It presents the general architectural model of Autonomic Computing Systems and it takes a detailed look into how Mutli-Agent methodologies can be adapted for two of the major paradigms of Autonomic Computing. The study closes with a presentation of some of applications of the Autonomic Computing field.

1. Introduction

Autonomic Computing advocates for the self-management of software systems, with the purpose of replacing humans in tasks of installation, maintenance, configuration, optimization and protection of such systems.

The need for Autonomic Computing systems arises from the increasing complexity of managing large scale interconnected computing environments [1]. These systems will become more and more massive and complex, requiring an increasing number of IT skilled professionals for controlling them. Autonomic Computing provides a solution to this complexity problem by delegating the management paradigms to the systems themselves. Humans will only give high-level objectives to autonomic computing systems, without explicitly giving information on how to achieve them.

The Autonomic Computing systems model has been inspired by the Human Nervous System, which has a complete autonomic behavior [2]. Control functions perform below the level of consciousness with the scope of influencing and regularizing specific anatomic functions. Examples of such functions are the control of respiration, cardiac regulation, coughing,

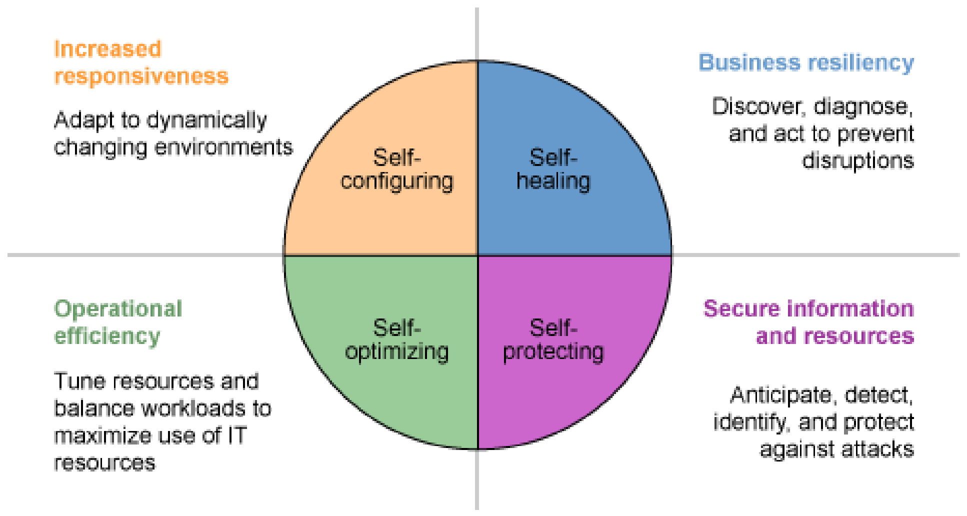
Increased responsiveness
Adapt to dynamically changing environments
Self-configuring
Self-healing
Business resiliency
Discover, diagnose, and act to prevent disruptions
Operational efficiency
Tune resources and balance workloads to maximize use of IT resources
Self-optimizing
Self-protecting
Secure information and resources
Anticipate, detect, identify, and protect against attacks

Figure 1: Self CHOP Paradigm (Self-Configuring; Healing; Optimizing; Protecting)
(Image reproduction from Miller [9])

swallowing and many others. The autonomic nervous system can be divided into sensory and motor subsystems. The sensory system is responsible for processing sensory information while the motor system is the one carrying the motor impulses to the voluntary muscles.

2. The Vision of Self-Management

In order to achieve Self-Management, complex software systems need to address a number of aspects, such as Self-Configuration, Self-Healing, Self-Optimization and Self-Protecting (abbreviated as CHOP, see figure 1). These characteristics will be treated independently and then combined into a general architecture under the general umbrella of self-maintenance.

If the vision of developing Self-Management systems is plotted on a timeline, we could distinguish some important milestones [1]. Initially, systems could collect and report information requested by human admins to support their decisions. At a second stage, the systems could suggest possible decision actions that the human admins can consider or not. In the third stage, humans could entrust autonomic systems with making low level decision, while high level decisions are still performed by humans. As the development of autonomous systems reaches the final stage, and the trust of automated decisions grows, humans will take for granted the computing system self-management process.

Self-Configuration

There is a need for complex distributed systems to have sub-parts configured, installed, integrated and deployed. Such tasks can have an increased complexity even for human experts, leading to time and financial costs. The configuration of such systems whenever a new data storage or processing node is being added or removed is an intricate process if solely done by humans.

On the other hand, Autonomic systems will be able to configure themselves in accordance to a set of rules. These rules will be expressed as high-level policies, representing business level objectives. Such policies will specify only what it is desired, and not how they are going to be accomplished by the system.

For example large data centers used by big companies such as Google or Microsoft contain thousands of computing nodes, on many different hardware or software platforms, covering databases or web-service related technologies. When a new processing node is introduced to the datacenter, the autonomic system will automatically learn about it and register it such that other processing nodes can either use it or modify their behavior in accordance to the latest processing addition.

Self-Healing

Software and hardware components are prone to bugs and failures. Using humans to root-trace and identify such issues in complex systems can be a very expensive and time consuming process.

Automatic Computing systems are able to detect and localize the potential issues. A diagnose phase can be run, analyzing the contextual information and matching the problem against actions or software patches. Ultimately the system will recover from the detected problems and will continue to function correctly.

For example, Microsoft uses a mechanism for diagnosing and solving automatically issues that appear on client machines in newer versions of Windows Operating Systems (Windows 7 and Windows 8). Information specific to the issue (like erroneous module, stack trace and memory snapshot) are uploaded to a centralized system. An analysis phase follows, during which the pattern of the issue (bucket) is determined. The client machine will receive the corresponding bucket fix, actions or steps that will resolute the problem. Due to this mechanism, Windows 7 was called the First “Self-Healing” Operating System [5].

Self-Optimization

Autonomic systems are able to improve their execution by optimizing themselves. They are able to detect sub-optimal behavior and learn to make appropriate choices.

For example large database systems such as Oracle, may have hundreds of tunable parameters [1]. In order for the system to perform optimally, all these parameters need to be set correctly. An autonomous system can decide on best parameter values by monitoring and experimenting. Also, the system can regularly inspect the versions of inner components and perform upgrades with latest updates.

Self-Protection

Firewall, antiviruses and intrusion detection tools are used widely for preventing internal and external malicious attacks in software systems. It is desired to move the decision factor on how to use the above tools from a manual human process into an automated self-aware and self-protective decision mechanism.

Autonomic computing systems are able to protect and detect from arbitrary attacks. Reports can be gathered proactively from different sensors and concrete steps will be implemented to resolute possible exploits.

For example, intelligent agents could be designed to learn what action or program executions can cause a problem to the system, prevent such attacks and heal the system once an attack has taken place [6]. Execution patterns that are seen many times are considered to be benign ones, while unseen executions are most likely anomalies, potentially caused by a malicious attack.

Context Aware

Autonomic computing systems are aware of their environment. If changes are happing in the environment, they should be able to react and adapt.

Open

Autonomic computing systems are built by using standard and open protocols and interfaces [2]. Hence, the system can to be ported to various software and hardware technologies and architectures.

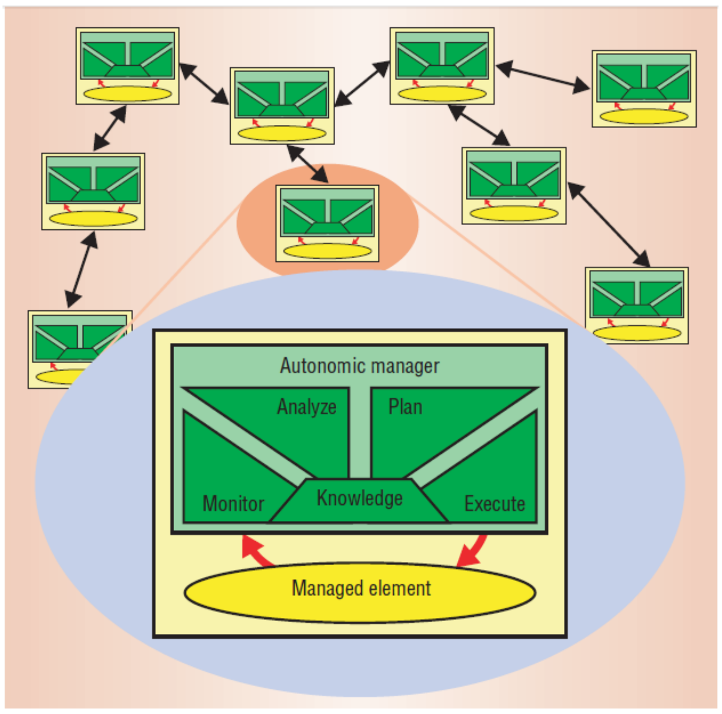

Figure 2: An interconnected set of Autonomic Elements. Each Autonomic Element consists of an Autonomic Manager and a Managed Element. (Image reproduction from: Kepher&Chess [1])

3. Architectural Model of Autonomic Computing

The architecture of an Autonomic Computing system consists of numerous interconnected Autonomic Elements. Figure 2 illustrates a network of such elements.

Autonomic Elements

Autonomic Elements are the core entity of the architectural model. These entities have the role of encapsulating a resource, called Managed Element, and provide interfaces or services to similar entities or humans. The self-managing process is being performed by the Autonomic Manager.

The functional goals of autonomic elements are dictated by humans or other autonomic elements through high level policies. The interactions between the elements are supported by a distributed Service Oriented Architecture [1].

Agent oriented architectures are critically important when designing an autonomic computing system. Within this perspective autonomic elements and the autonomic system correspond to agents respectively multi agent systems.

The Managed Element

Managed elements are the functional units and resources which need to be administered. Examples of such units are programs, databases, CPUs or printers. There can be multiple such elements within the same Autonomic Element.

The Managed elements receive input through Sensors Attached to the Monitoring component of the Autonomic Manager, and it outputs actions specified by the Execution component.

The Environment

The changes within the internal and external environments can impact the management element. The managed element and the environment can be seen as two sub-systems forming a stable system [2].

The Autonomic Manager

The Autonomic Manager controls and represents the managed elements. There are a number of functional responsibilities of Autonomic Managers, performed continually during the life-cycle of the Autonomic Element [2]:

1. Receives and interprets requirements specified by users through high level policies.
2. Queries the state of the Managed Element.
3. Characterizes the state of the entire distributed system of Autonomic Elements and the state of the environment.
4. Controls the operation of the Managed Element by using the acquired state information, with the objective of achieving the specified behavior.

The MAPE-K Autonomic Loop

The monitoring, analysis, plan, execution and knowledge functions of the autonomic manager are inner components of a control loop, which is sometimes referred to as the MAPE-K Autonomic Loop [7].

The monitoring component is responsible for capturing proprieties of the environment through various sensors. We can distinguish between Passive and Active monitoring. In passive monitoring the sensors observe the required data from outside without intervening. In active monitoring, the observable software entities are implemented in such a way that they report monitoring activity to subscribers.

The planning component considers the monitored data and produces a plan of actions to be executed on the managed element. The simplest implementation of the planning component is by defining event-condition-actions [7] that will produce plans for various event combinations.

The knowledge within the autonomic system can have various sources, like human experts or logs with accumulated and aggregated data storing all the system's operations and behavior. These logs can then be used for training predictive models. Methods for representing knowledge are reinforcement learning, the utility concept and Bayesian techniques [7].

4. Multi-Agent Systems Approach to Autonomic Computing

Autonomous agents can be used for the development of scalable and robust software systems [11]. They can achieve specific objectives even if situated in a dynamic and uncertain environment. Autonomic computing can follow a Multi-Agent System (MAS) approach to battle the complexity of increasingly large scale distributed systems. Automatic group formation, emergent behavior, multiagent adaptation and agent coordination can be successfully adapted from Multi-Agent Systems to the Autonomic Computing paradigm [10].

The Multi-Agent methodologies presented within this Section have been developed into a concrete prototype implementation by IBM (Unity [10]), which handles realistic Web-based transactional workloads running on a Linux cluster. The scope of Unity system is to self-assemble, recover from certain classes of failures and to manage the use of computational resources (such as servers) in a dynamic multi-application environment [10].

We will describe two of the self-management characteristics that can be modeled by using Mulit-agent inspired methodologies: Self-Configuration and Self-Healing. Before that, we introduce a number of new structural entities which are required.

Structural Entities

The multi-agent architecture uses the general model detailed in Section 3 as a starting point. Individual Autonomic elements that encapsulate Managed elements and Autonomic Managers are at the core of the design. Besides, an additional set of entities, most of them inspired from Multi-Agent systems is introduced:

Application environments	Each application service will be encapsulated in an application environment. There can exists multiple, logically separated application environments.
Application manager	The application manager is controlling the application environments in order to obtain required resources to meet the imposed goals. It also communicates with other elements on management matters. There is one application manager element for each application environment.
Resource arbiter	The computation for resource allocation and distributions to various application environments is performed by the Resource arbiter.
Server	The servers are the resources allocated within the system, corresponding to the Managed Elements in the architectural model. The role of the servers is to announce its address and usage capabilities.
OSContainer	Autonomic elements are hosted within computers called OSContainers. Their role is to receive requests for starting up services or autonomic elements.
Registry elements	Elements will locate other elements with which they need to interact by using the Registry elements.
Policy repository elements	Humans will introduce high-level policies by using interfaces of Policy repository elements.
Sentinel elements	Elements will request the monitoring of other elements by using interfaces exposed by Sentinel elements.

Goal-driven Self Assembly

Self-configuration within the multi-agent system can be achieved by goal-driven self-assemblies. During the initialization phase, only high level description of goals are known by the autonomous elements.

The following sequence is to be executed during the initialization phase by each element:

1. The element contacts the registry in order to locate other existing elements that can provide required services.
2. A relationship will be established with each element located at Step 1, in order to obtain required services.
3. Once all the relationships at Step 2 are established, and all the corresponding resources are obtained, the element will register itself with the registry. This means that other elements can locate it if its services are required.

Once initialized, the element needs to locate and contact the policy repository. This repository contains all the information that the element needs to know, besides the registry address and its own high-level objectives. Configuration decisions are drafted from the information retrieved from the policy repository.

It follows a bootstrap process, which has the following steps:

1. OSContainers and the registry start, since they are necessary for starting all the other elements.
2. Resource arbiter starts, then decides what other elements need to be started. Starting future elements will be arranged by communicating with the OSContainers.
3. The Resource arbiter registers with the Registry. It locates then the existing policy repository and sentinels and asks a sentinel to monitor each Policy Repository. Note that Policy repository and Sentinels register with the registry immediately after initialization.
4. Server elements get in touch with the Resource arbiter, in order to announce themselves.
5. Application environments managers query the Resource arbiter for servers which they can use.

It can be noted that by following the proposed sequence, the elements don't need to know beforehand where other elements are located, and how many elements of a specific type will exist.

Self-Healing For Clusters

In order to achieve self-healing within imperfect software and hardware under-layers, the policy repository can join a Synchronized policy repository Cluster. All the data changes within the policy repository will be replicated within that cluster. The second function of the cluster is to detect the failure of one of its members and come up with a replacement.

The operation of constructing the self-healing synchronized cluster has the following phases:

1. During system initialization, the Resource arbiter determines the set of all Policy repositories that will be part of the system.
2. Resource arbiter deploys all the policy repositories found during step 1, each being also supplied with the identifier of the cluster it will have to join.
3. During initialization of each policy repository, the registry is consulted on already registered members of the associated cluster identifier, and thus joining the cluster.
4. The resource arbiter sets-up a sentinel that will monitor each of these policy repositories.

If any policy repository receives a request to change its policy set, the rest of the cluster will be announced about the required updates.

If one policy repository fails, the sentinel will notify the resource arbiter. The resource arbiter will then choose one cluster member which is still functioning within correct parameters. It will assign to the new policy repository the subscriptions handled by the failed member. Then, it will notify all the cluster members of the re-assignments.

The described model contains a central point of failure, namely the sole sentinel [10]. To solve this issue, a cluster of sentinels can be introduced similarly to the process described for policy repositories.

Autonomic Computing Applications

Wireless Sensor Networks are benefiting from the adoption of Autonomic Computing. Routing needs to be performed with minimal delivery time and minimal consumed energy. By using an autonomic computing approach, routing can find a near optimal route without knowing the network topology [7].

Autonomic computing has also been applied in Data Centers, Clusters and GRID Computing Systems. Maintaining a geographically distributed system containing high-performance clusters of computers can be solved by using Autonomic computing. Dynamic resource management and systems administration can be modeled through Autonomic Computing Paradigms such that the system provides an agreed quality of service.

Within the IT industry, IBM Research Autonomic Computing group [1] is one of the biggest players in this field. Microsoft constructed the Dynamic Systems Initiative, with the scope of simplifying and automating how businesses design, deploy, and operate distributed systems [12]. Other notable Autonomic Computing research centers within the IT industry are: Planetary Computing (HP Labs), Dynamic Computing Initiative (Dell) and Harmonious Computing (Hitachi).

Conclusion

Autonomic computing can provide solutions in a number of application domains comprising complex, distributed systems. They are inspired from the human Autonomic Nervous System and provide self-managing characteristics, such as: self-configuration, self-healing, self-optimization and self-protection. Multi-Agent strategies and algorithms can successfully be used to model the architecture of Autonomous Computing Systems.

References

[1] Kephart, J., Chess, D.: The Vision of Autonomic Computing, January 2003, IBM Thomas J. Watson Research Center

[2] Parashar, M., Hariri, S.: Autonomic Computing: An Overview. University of Arizona, High Performance Distributed Computing Laboratory.

[3] IBM Corporation: An architectural blueprint for autonomic computing. April 2003.

[4] Nami, M., Bertel, K.: A Survey of Autonomic Computing Systems. Delft University of Technology, Computer Engineering Laboratory.

[5] McDougall, P.: Windows 7 First “Self Healing” OS, in Information Week 10/30/2009.

[6] Fernandez, L., Terashima, H., Valuenzuela, M.: An Autonomic Computing Mechanism to Enable Self-Protection and Self-Healing. Technical Institute of Monterey, Center for Intelligent Computing and Robots.

[7] Huebscher, M., McCann J.: A survey of autonomic computing – degrees, models, applications. Imperial College London.

[8] Sterritt,R. : Autonomic Computing, in Innovations Syst Softw Eng (2005) 1: 79–88

[9] Miller, B. : The autonomic computing edge: Can you CHOP up autonomic computing?, IBM Developer Works, 03/19/2008.

[10] Tesauro, G., Chess, D., Walsh, W., Das, R., Segal, A., Whalley, I., Kephart, J., White, S.: Multi Agent Systems Approach to Autonomic Computing. IBM Research Division.

[11] Jennings, N.: On agent-based software engineering. Artificial Intelligence 117 (2000) 277–296

[12] Microsoft Corporation: Microsoft Dynamic Systems Initiative Overview, March 2004.