IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 15, Issue 1 (Sep. - Oct. 2013), PP 35-41
www.iosrjournals.org

Decision Making and Autonomic Computing

Rajyagor Bhargav P.
Affiliated to: Gujarat Technological University MCA Department – SBIC (Chaparda)

Abstract: *Autonomic Computing refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users.*

An autonomic system makes decisions on its own, using high-level policies; it will constantly check and optimize its status and automatically adapt itself to changing conditions. As widely reported in literature, an autonomic computing framework might be seen composed by autonomic components interacting with each other.

An Autonomic Computing can be modeled in terms of two main control loops (local and global) with sensors (for self-monitoring), effectors (for self-adjustment), knowledge and planner/adapter for exploiting policies based on self- and environment awareness.

The goal of autonomic computing is to create systems that run themselves, capable of high-level functioning while keeping the system's complexity invisible to the user.

General Terms: *Autonomic systems, Self-configuration, Self-healing, Self-optimization, Self-protection.*
Keywords: *Know itself, reconfigure, recover from extraordinary events, expert in self-protection,*

I. Introduction

The term autonomic computing is representative of a vast and somewhat twisted hierarchy of natural self-governing systems, many of which consist of countless interacting, self-governing components that in turn comprise large numbers of interacting, autonomous, self-governing components at the next level down.

The term autonomic is derived from human biology. The autonomic nervous system monitors your heartbeat, checks your blood sugar level and keeps your body temperature close to 98.6°F, without any conscious effort on your part.

In much the same way, autonomic computing components anticipate computer system needs and resolve problems—with minimal human intervention. However, there is an important distinction between autonomic activity in the human body and autonomic responses in computer systems.

Many of the decisions made by autonomic elements in the body are involuntary, whereas autonomic elements in computer systems make decisions based on tasks you choose to delegate to the technology. In other words, adaptable policy— rather than rigid hard coding—determines the types of decisions and actions autonomic elements make in computer systems.

Autonomic computing can result in a significant improvement in system management efficiency, when the disparate technologies that manage the environment work together to deliver performance results system wide.

The journey toward fully autonomic computing will take many years, but there are several important and valuable milestones along the path.

At first, automated functions will merely collect and aggregate information to support decisions by human administrators.

Later, they will serve as advisors, suggesting possible courses of action for humans to consider.

As automation technologies improve, and our faith in them grows, we will entrust autonomic systems with making—and acting on—lower-level decisions.

Over time, humans will need to make relatively less frequent predominantly higher-level decisions, which the system will carry out automatically via more numerous, lower- level decisions and actions.

II. Autonomic Computing

An autonomic computing system consists of eight key characteristics as follows:

- To be autonomic, a computing systems needs to “know itself”-and comprise components that also possess a system identity
- An autonomic computing system must configure and reconfigure itself under varying and unpredictable conditions.
- An autonomic computing system never settles for the status quo-it always looks for ways to optimize its workings.

- An autonomic computing system must perform something akin to healing-it must be able to recover from routine and extraordinary events that might cause some of its parts to malfunction.
- A virtual world is no less dangerous s than the physical one, so an autonomic computing system must be an expert in self-protection
- An autonomic computing system knows its environment and the context surrounding its activity, and acts accordingly
- An autonomic computing system cannot exist in a hermetic (protected from outside influence) environment.
- Perhaps most critical for the user, an autonomic computing system will anticipate the optimized resources needed while keeping its complexity hidden

III. Architectural Considerations

The autonomic computing architecture concepts provide a mechanism for discussing; comparing and contrasting the approaches different vendors use to deliver self-managing attributes in an autonomic computing system.

The autonomic computing architecture starts from the premise that implementing self-managing attributes involves an intelligent control loop. This loop collects information from the system, makes decisions and then adjusts the system as necessary.

An intelligent control loop can enable the system to do such things as:

- **Self-configuration:** Automatic configuration of components;
- **Self-healing:** Automatic discovery, and correction of faults;
- **Self-optimization:** Automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements;
- **Self-protection:** Proactive identification and protection from arbitrary attacks.

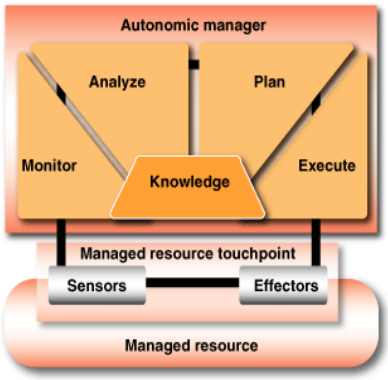


Figure: Autonomic Computing Architecture

3.1 Autonomic manager internal structure

The autonomic computing architecture does not prescribe the specific implementation choices for the internal structure of an autonomic manager. However, the architecture does organize the internal structure into a set of capabilities or functions. These are illustrated in Figure 4 and described in the following sections.

Monitor

The monitor function collects the details from the managed resources, via touchpoints, and correlates them into symptoms that can be analyzed.

The details can include topology information, metrics, configuration property settings and so on. This data includes information about managed resource configuration, status, offered capacity and throughput. Some of the data is static or changes slowly, whereas other data is dynamic, changing continuously through time.

The monitor function aggregates, correlates and filters these details until it determines a symptom that needs to be analyzed. For example, the monitor function could aggregate and correlate the content of events received from multiple resources to determine a symptom that relates to that particular combination of events. Logically, this symptom is passed to the analyze function.

Autonomic managers must collect and process large amounts of data from the touchpoints sensor interface of a managed resource (detailed in the Managed resource section).

An autonomic manager's ability to rapidly organize and make sense of this data is crucial to its successful operation.

Analyze

The analyze function provides the mechanisms to observe and analyze situations to determine if some change needs to be made. For example, the requirement to enact a change may occur when the analyze function determines that some policy is not being met.

The analyze function is responsible for determining if the autonomic manager can abide by the established policy, now and in the future. In many cases, the analyze function models complex behavior so it can employ prediction techniques

Autonomic managers must be able to perform complex data analysis and reasoning on the symptoms provided by the monitor function. The analysis is influenced by stored knowledge data, described later.

If changes are required, the analyze function generates a change request and logically passes that change request to the plan function. The change request describes the modifications that the analyze component deems necessary or desirable.

Plan

The plan function creates or selects a procedure to enact a desired alteration in the managed resource. The plan function can take on many forms, ranging from a single command to a complex workflow.

The plan function generates the appropriate change plan, which represents a desired set of changes for the managed resource, and logically passes that change plan to the execute function.

Execute

The execute function provides the mechanism to schedule and perform the necessary changes to the system. Once an autonomic manager has generated a change plan that corresponds to a change request, some actions may need to be taken to modify the state of one or more managed resources.

The execute function of an autonomic manager is responsible for carrying out the procedure that was generated by the plan function of the autonomic manager through a series of actions. These actions are performed using the touch point effector interface (detailed later) of a managed resource.

Part of the execution of the change plan could involve updating the knowledge that is used by the autonomic manager (described next).

Knowledge Source

A knowledge source is an implementation of a registry, dictionary, database or other repository that provides access to knowledge according to the interfaces prescribed by the architecture. In an autonomic system, knowledge consists of particular types of data with architected syntax and semantics, such as symptoms, policies, change requests and change plans. This knowledge can be stored in a knowledge source so that it can be shared among autonomic managers.

The knowledge stored in knowledge sources can be used to extend the knowledge capabilities of an autonomic manager.

An autonomic manager can load knowledge from one or more knowledge sources, and the autonomic manager's manager can activate that knowledge, allowing the autonomic manager to perform additional management tasks (such as recognizing particular symptoms or applying certain policies).

Knowledge Data used by the autonomic manager's four functions (monitor, analyze, plan and execute) are stored as shared knowledge. The shared knowledge includes data such as topology information, historical logs, metrics, symptoms and policies.

Knowledge - Standard data shared among the monitor, analyze, plan and execute functions of an autonomic manager, such as symptoms and policies

Knowledge types

The Autonomic Computing blueprint identifies several types of system knowledge. These include solution topology knowledge, policy knowledge, and problem determination knowledge scenarios. Table 1 summarizes various types of knowledge that may be present in a self-managing autonomic system. Each knowledge type must be expressed using common syntax and semantics so the knowledge can be shared.

Solution Topology Knowledge

Captures knowledge about the components and their construction and configuration for a solution or business system.

Installation and configuration knowledge is captured in a common installable unit format to eliminate complexity. The plan function of an autonomic manager can use this knowledge for installation and configuration planning.

Policy Knowledge

A policy is knowledge that is consulted to determine whether or not changes need to be made in the system. An autonomic computing system requires a uniform method for defining the policies that govern the decision-making for autonomic managers. By defining policies in a standard way, they can be shared across autonomic managers to enable entire systems to be managed by a common set of policies.

Problem Determination Knowledge

Problem determination knowledge includes monitored data, symptoms and decision trees. The problem determination process also may create knowledge. As the system responds to actions taken to correct problems, learned knowledge can be collected within the autonomic manager. An autonomic computing system requires a uniform method for representing problem determination knowledge, such as monitored data (common base events), symptoms and decision trees.

Touchpoints

A touchpoint is the component in a system that exposes the state and management operations for a resource in the system. An autonomic manager communicates with a touchpoint through the manageability interface, described next. A touchpoint, depicted in Figure 5, is the implementation of the manageability interface for a specific manageable resource or a set of related manageable resources. For example, there might be a touchpoint implemented that exposes the manageability for a database server, the databases that database server hosts, and the tables within those databases.

A sensor consists of one or both of the following:

• A set of properties that expose information about the current state of a manageable resource and are accessed through standard "get" operations.

• A set of management events (unsolicited, asynchronous messages or notifications) that occur when the manageable resource undergoes state changes that merit reporting

These two parts of a sensor interface are referred to as interaction styles. The "get" operations use the request-response interaction style; events use the send-notification interaction style.

An effector consists of one or both of the following:

• A collection of "set" operations that allow the state of the manageable resource to be changed in some way

• A collection of operations that are implemented by autonomic managers that allow the manageable resource to make requests from its manager.

The "set" operations use the perform-operation interaction style; requests use the solicit-response interaction style to allow the manageable resource to consult with its manager.

The sensor and effector in the architecture are linked together. For example, a configuration change that occurs through the effector should be reflected as a configuration change notification through the sensor interface. The linkage between the sensor and effector is more formally defined using the concept of manageability capabilities.

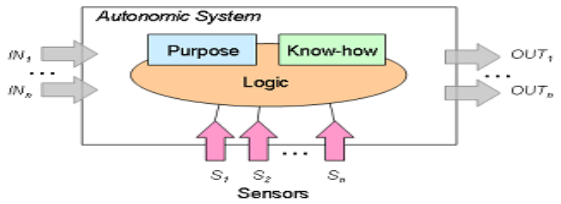
A manageability capability refers to a logical collection of manageable resource state information and operations.

3.2 Conceptual model of autonomic system

A fundamental building block of an autonomic system is the sensing capability (Sensors S), which enables the system to observe its external operational context. Inherent to an autonomic system is the knowledge of the Purpose (intention) and the Know-how to operate itself (e.g., bootstrapping, configuration knowledge, interpretation of sensory data, etc.) without external intervention.

The actual operation of the autonomic system is dictated by the Logic, which is responsible for making the right decisions to serve its Purpose, and influence by the observation of the operational context (based on the sensor input).

This model highlights the fact that the operation of an autonomic system is purpose-driven. This includes its mission (e.g., the service it is supposed to offer), the policies (e.g., that define the basic behavior), and the survival instinct. If seen as a control system this would be encoded as a feedback error function or in a heuristically assisted system as an algorithm combined with set of heuristics bounding its operational space.



3.3 Autonomic element

The building block of autonomic computing system is called an autonomic element. An autonomic element is an individual system constituent that contains resources and delivers services to human and other autonomic elements.

Autonomic elements learn from past experiences by managing their internal behavior and their relationship with other autonomic elements in accordance with guidelines that humans or other elements have established to manufacture and execute action plans.

The formation of an autonomic element consists of an autonomic manger and managed elements. According to An architectural blueprint for autonomic computing, a managed element is what the autonomic manager is controlling, and an autonomic manager is component that implements a particular control loop.

IV. DECISION MAKING FOR AUTONOMIC COMPUTING

Autonomic Computing refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users.

An autonomic system makes decisions on its own, using high-level policies; it will constantly check and optimize its status and automatically adapt itself to changing conditions. As widely reported in literature, an autonomic computing framework might be seen composed by autonomic components interacting with each other.

Driven by such vision, a variety of architectural frameworks based on self-regulating autonomic components has been recently proposed. A very similar trend has recently characterized significant research in the area of multi-agent systems. However, most of these approaches are typically conceived with centralized or cluster-based server architectures in mind and mostly address the need of reducing management costs rather than the need of enabling complex software systems or providing innovative services.

Autonomic systems

A possible solution could be to enable modern, networked computing systems to manage themselves without direct human intervention. The Autonomic Computing Initiative (ACI) aims at providing the foundation for autonomic systems. It is inspired by the autonomic nervous system of the human body.

In a self-managing autonomic system, the human operator takes on a new role: instead of controlling the system directly, he/she defines general policies and rules that guide the self-management process. For this process, IBM defined the following four functional areas:

- Self-configuration
- Self-healing
- Self-optimization
- Self-protection

IBM defined five evolutionary levels, or the autonomic deployment model, for its deployment:

This evolutionary path to autonomic computing is represented by five levels, starting from basic, through managed, predictive, adaptive and finally to autonomic.

- The basic level represents the starting point where a significant number of IT systems are today. Each element of the system is managed independently by systems administrators who set it up, monitor it, and enhance it as needed.
- At the managed level, systems management technologies are used to collect information from disparate systems into one, consolidated view, reducing the time it takes for the administrator to collect and synthesize information.
- At the predictive level, new technologies are introduced that provide correlation among several elements of the system. The system itself can begin to recognize patterns, predict the optimal configuration and provide advice on what course of action the administrator should take. As these technologies improve, people will become more comfortable with the advice and predictive power of the system.

- The adaptive level is reached when systems can not only provide advice on actions, but can automatically take the right actions based on the information that is available to them on what is happening in the system.
- Finally, the full autonomic level would be attained when the system operation is governed by business policies and objectives. Users interact with the system to monitor the business processes, and/or alter the objectives.

V. Fault Model Characteristics

The Following are typical fault model characteristics that seem relevant:

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: Naturally, 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 Calculating faults which are considered "arbitrary" faults. 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 so on.

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.

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.

VI. Benefits And Future Application Scope Of Autonomic Applications

There are short term and long term benefits of autonomic systems. The *sort terms benefits* are in terms of cost saving as autonomic systems are easy to scale up. The idle processing can be use utilized fully through networked systems. The systems will be highly available, stable as there will be lesser error as systems will be self-healing. The *long term benefits* will be to achieve end to end service management. Another long term benefit will be to embed autonomic capabilities in client or access devices, servers, storage systems, middle-ware and network itself to make them more robust.

Autonomic computing can be implemented in various areas like electricity, railway management, air traffic control systems etc. Few of the possible application areas are discussed as follows:

Electricity Transmission Systems:

Current electricity transmission systems employ manual management of interconnected subsystems that supply electricity to the users. This manual management is inefficient and error prone. To remove these shortcomings, autonomic capability can be integrated to these systems. The sensor could monitor the environment factors such as temperature, humidity, wind speed, month, etc. and autonomously provide the necessary power. Any short- circuit could also be detected and as a measure to correct it supply is halted in that node immediately and a worker is notified to for error correction. This would greatly reduce the probability of power-cuts and dynamically provide electricity as required. Self-optimization can be thought of in terms of providing load forecasting facility on the basis of pervious electricity requirements. System will also provide security against threats such as unauthorized accesses.

Train Route Management System:

Introduction of high-speed trains has posed the need for a faster management of the whole system. Manual management is quite slow as compared to management through computers. Also the level of imprecision in these real time systems is highly dynamic. Train schedules are also dynamic. Autonomic computing can provide this speed of decision making. This technique of using technology to manage technology has real benefits to these real time systems. The system will configure itself upon introduction of new trains and new tracks. This will also decrease collision probability or any slowdown due to failure at any point. System will minimize cost and time of service providing efficient service in a convenient way. This system will too provide security against threats such as unauthorized accesses.

Multiprocessor System:

Both versions of multiprocessing i.e. symmetric as well as asymmetric are hard-coded and have their advantages and shortcomings also. The processors can be configured to process their specific processes. But this account for performance slowdown when there is load on only one processor. This provides the motivation to find a way to manage the processor load autonomously. Self-management capability is the key to successful multiprocessing where each processor is running their own instance of operating system. With autonomic capability the throughput, resource utilization, performance of the system will increase.

Traffic Control System:

Centralized architecture of current traffic control systems will soon become brittle and unmanageable as the traffic volume and number of devices used as a sensor to provide situation of traffic at a time increases. Thus, there is a need of a decentralized system that consists of autonomic devices capable of monitoring the environment and communicates with other adjacent autonomic devices so as to make decisions for the complete traffic management. Self- configuration nature can be built as the capability to identify new node and take it into working use. If a node fails others around must notify and take charge of its working. This system will also provide users with current traffic condition so that they can reach their destination conveniently.

Travel Guide:

A Travel-Guide application can be developed which is capable to provide location-dependent service in response to a change in the user's location. GPS system can be used to track a user and provide this input to an autonomic element which can then search for available services in that locality in its database.

VII. Conclusion

Autonomic computing offers as least as many benefits in the security area as it does challenges. The complexity of modern computing systems makes secure systems administration a daunting task and one that is seldom done well in practice.

Recent advances, including the growing use of automatic intrusion detection systems, secure embedded processors, proactive security measures, and automated virus response, have helped take some of the burden of security maintenance off overloaded system administrators, but there is much more to do. By making computing systems directly aware of the security policies that apply to them, and giving the systems the ability to conform their actions to those policies, the techniques of autonomic computing will help create systems that are increasingly and consistently secure.

This new view of computing will necessitate changing the industry's focus on processing speed and storage to one of developing distributed networks that are largely self-managing, self-diagnostic, and transparent to the user.

References

[1] IBM, "Autonomic Computing: IBM's Perspective on the State of Information Technology"; http://www.research.ibm.com/autonomic/
[2] IBM White Paper, "An architectural blueprint for autonomic computing", IBM
[3] Sterritt, Roy and M. Hinchey, "Tutorial proposal: autonomic computing in real-time systems".
[4] IBM, "Autonomic Computing: IBM's Perspective on the State of Information Technology"; http://www-1. ibm.com/industries/government/doc/content/resource/ thought/278606109.html.
[5] Abdullah, Gani and G. Manson, "Towards a Self-Healing Networking Controlling Access to Network Applications", Informing Science.
[6] Philip Koopman, "Elements of the Self-Healing System Problem Space", ICSE
[7] http://searchcio-midmarket.techtarget.com/definition/autonomic-computing
[8] Journal of Global Research in Computer Science, 3 (5), May 2012