An architectural blueprint for autonomic computing.
## Contents

1. Introduction  
   - Autonomic computing  
     - Self-management attributes of system components  
     - Delegated IT processes can deliver self-managing capabilities  
   - Customer value  
   - Motivation for a blueprint  

2. Autonomic computing architecture concepts  
   - Autonomic computing system  
   - Managed resource  
   - Touchpoints  
   - Touchpoint autonomic managers  
   - Orchestrating autonomic managers  
   - Integrated solutions console  
   - Autonomic computing blueprint illustration  

3. Autonomic computing architecture details  
   - Autonomic manager  
     - Autonomic manager internal structure  
       - Monitor function  
       - Analyze function  
       - Plan function  
       - Execute function  
       - Knowledge  
       - Knowledge Types  
   - Managed resource  
   - Manageability Interface  
     - Sensor  
     - Effector  

4. Evolving maturity and sophistication  
   - How the architecture supports autonomic computing evolution  

5. Standards for autonomic computing  

6. Summary
1. Introduction

In an on demand business, IT professionals must strengthen the responsiveness and resiliency of service delivery—improving quality of service while reducing the total cost of ownership (TCO) of their operating environments. Yet, information technology (IT) components produced by high-tech companies over the past decades are so complex that IT professionals are challenged to effectively operate a stable IT infrastructure. It’s the complexity of the IT components themselves that have helped fuel this problem. As networks and distributed systems grow and change, system deployment failures, hardware and software issues, and human error can increasingly hamper effective systems administration. Human intervention is required to enhance the performance and capacity of the components in an IT system, driving up overall costs—even as technology component costs continue to decline.

We do not see a slowdown in Moore’s law as the main obstacle to further progress in the IT industry. Rather, it is the industry’s exploitation of the technologies that have been developed in the wake of Moore’s law that has led us to the verge of a complexity crisis. Software developers have fully exploited a 4-to-6 order-of-magnitude increase in computational power—producing ever more sophisticated software applications and environments. Exponential growth has occurred in the number and variety of systems and components. The value of database technology and the Internet have fueled significant growth in storage subsystems, which now are capable of holding petabytes of structured and unstructured information. Networks have interconnected distributed, heterogeneous systems. Our information society has created unpredictable and highly variable workloads for these networked systems. And these increasingly valuable, complex systems require highly skilled IT professionals to install, configure, operate, tune and maintain them.

Autonomic computing

Autonomic computing helps address this complexity by using technology to manage technology. 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 capabilities anticipate IT system requirements and resolve problems, with minimal human intervention. As a result, IT professionals can focus on tasks with higher value to the business.

However, there is an important distinction between autonomic activity in the human body and autonomic activities in IT systems. Many of the decisions made by autonomic capabilities in the body are involuntary. In contrast,
autonomic capabilities in computer systems make decisions based on tasks that IT professionals choose to delegate to the technology according to policies. Adaptable policy—rather than hard-coded procedure—determines the types of decisions and actions that autonomic capabilities perform.

Autonomic capabilities in a system accomplish their functions by taking an appropriate action based on one or more situations that they sense in the environment. The function of any autonomic capability is a control loop that collects details from the system and acts accordingly. Although there can be numerous types of control loops in a system, this paper organizes these control loops into four categories: self-configuring, self-healing, self-optimizing, and self-protecting. This paper also describes two fundamental ways in which the control loops for these functions can be constructed in an IT infrastructure (within an autonomic manager or embedded within a managed resource).

Policy - A set of considerations that are designed to guide the decisions that affect the behavior of a managed resource task. These environments are self-configuring, self-healing, self-optimizing, and self-protecting.

Self-Configure - To adapt to dynamically changing environments.

Self-Heal - To discover, diagnose and act to prevent disruptions.

Self-management attributes of system components

In an autonomic computing environment, system components—from hardware (such as storage units, desktop computers and servers) to software (such as operating systems, middleware and business applications)—can include embedded control loop functionality. Although these control loops consist of the same fundamental parts, their functions can be divided into four broad embedded control loop categories. These categories are considered to be attributes of the system components and are defined as:

- **Self-configuring**—Can dynamically adapt to changing environments
  Self-configuring components adapt dynamically to changes in the environment, using policies provided by the IT professional. Such changes could include the deployment of new components or the removal of existing ones, or dramatic changes in the system characteristics. Dynamic adaptation helps ensure continuous strength and productivity of the IT infrastructure, resulting in business growth and flexibility.

- **Self-healing**—Can discover, diagnose and react to disruptions
  Self-healing components can detect system malfunctions and initiate policy-based corrective action without disrupting the IT environment. Corrective action could involve a product altering its own state or effecting changes in other components in the environment. The IT system as a whole becomes more resilient because day-to-day operations are less likely to fail.
Self-Optimize - To tune resources and balance workloads to maximize the use of information technology resources.

Self-Protect - To anticipate, detect, identify and protect against threats.

Change Management - The process of planning (for example, scheduling) and controlling (for example, distributing, installing and tracking) software changes over a network.

• Self-Optimizing—Can monitor and tune resources automatically
  Self-optimizing components can tune themselves to meet end-user or business needs. The tuning actions could mean reallocating resources—such as in response to dynamically changing workloads—to improve overall utilization, or ensuring that particular business transactions can be completed in a timely fashion. Self-optimization helps provide a high standard of service for both the system’s end users and a business’s customers.

Without self-optimizing functions, there is no easy way to divert excess server capacity to lower priority work when an application does not fully use its assigned computing resources. In such cases, customers must buy and maintain a separate infrastructure for each application to meet that application’s most demanding computing needs.

• Self-Protecting—Can anticipate, detect, identify and protect against threats.
  Self-protecting components can detect hostile behaviors as they occur and take corrective actions to make themselves less vulnerable. The hostile behaviors can include unauthorized access and use, virus infection and proliferation, and denial-of-service attacks. Self-protecting capabilities allow businesses to consistently enforce security and privacy policies.

When system components have these attributes, it is possible to automate the tasks that IT professionals must perform today to configure, heal, optimize and protect the IT infrastructure.

Delegated IT processes can deliver self-managing capabilities
Self-managing system components can make adjustments only within their own scope. For example, a self-optimizing autonomic manager dedicated to a single server can optimize only that server’s operation. However, this is not the only area within an IT environment that self-managing capabilities can exist. The tasks associated with control loops that configure, heal, optimize and protect also can be found in the best practices and processes used to operate an IT organization.

IT businesses organize these tasks as a collection of best practices and processes such as those defined in the IT Infrastructure Library (from the Office of Government Commerce in the United Kingdom) and the IBM® IT Process Model (developed by IBM Global Services). Figure 1 (on page 6) shows some example process flows for incident management, problem management and change management. The more these tasks can be automated, the more opportunity for IT professionals to delegate the management of the IT infrastructure to itself.
The actual implementations of these processes in a particular IT organization vary, but their goals and functions are similar. It is possible to categorize the activities for these processes into four common functions: collect the details to identify a need, analyze the details to determine what should be done to fulfill the need, create a plan to meet the need, and execute that plan. For the system itself to manage these processes, the following conditions must exist:

1. The tasks involved in configuring, healing, optimizing, and protecting the IT system need to be automated.
2. It must be possible to initiate these processes based on situations that can be observed or detected in the IT infrastructure.

When these conditions exist in the IT infrastructure, IT professionals can configure the automated functions in a set of composed IT processes to allow the IT system to manage itself. These autonomic computing capabilities typically are delivered as management tools or products.
Customer value

The efficiency and effectiveness of typical IT processes are measured using metrics such as elapsed time to complete a process, percentage executed correctly and the cost to execute a process. Autonomic systems can positively affect these metrics, improving responsiveness and quality of service, reducing TCO and enhancing time to value through:

• Rapid process initiation—Typically, implementing these processes requires an IT professional to initiate the process, create the request for change, collect incident details and open a problem record. In a self-managing system, components can initiate these processes based on information derived directly from the system. This helps reduce the manual labor and time required to respond to critical situations, resulting in two immediate benefits: more timely initiation of the process and more accurate data from the system.

• Reduced time and skill requirements—These processes include tasks or activities that are skill-intensive, long lasting and difficult to complete correctly because of system complexity. In a change management process, one such activity is the “assess change impact” task. In a problem management process, one such activity is the “diagnose problem” task. In self-managing systems, resources are created such that the expertise required to perform these tasks can be encoded within the system and the task can be automated. This helps to reduce the amount of time and the degree of skill required to perform these tedious tasks. Hence, IT professionals are freed to perform higher value tasks, such as establishing business policies that the IT system needs to fulfill.

These intuitive and collaborative characteristics of the self-management capabilities enable businesses (large enterprises as well as small and medium-sized companies) to operate their business processes and IT infrastructures more efficiently with less human intervention, decreasing costs and enhancing the organization’s ability to react to change. For instance, a self-managing system could simply deploy a new resource and then tune the environment to optimize the services delivered by the new resource. This is a notable shift from traditional processes that require a significant amount of analysis before and after deployment to ensure that the resource operates effectively and efficiently.

Motivation for a blueprint

The idea of using technology to manage technology is not new—many companies in the IT industry have developed and delivered products based on this concept. Autonomic computing can result in a significant improvement in system management efficiency. However this is only possible when the disparate technologies that manage the IT environment work together to deliver perfor-
Highlights

**Autonomic Computing System** - A computing system that senses its operating environment, models its behavior in that environment, and takes action to change the environment or its behavior. An autonomic computing system has the properties of self-configuration, self-healing, self-optimization and self-protection.

**Managed Resource** - An entity that exists in the run-time environment of an IT system and that can be managed.

**Touchpoint** - The interface to an instance of a managed resource, such as an operating system or a server. A touchpoint implements sensor and effector behavior for the managed resource, and maps the sensor and effector interfaces to existing interfaces.

**Autonomic Manager** - A component that manages other software or hardware components using a control loop. The control loop of the autonomic manager includes monitor, analyze, plan and execute functions.

**Touchpoint Autonomic Manager** - An autonomic manager that works with managed resources through their touchpoints.

This architectural blueprint for autonomic computing is an overview of the fundamental concepts, constructs and behaviors for building autonomic capability into an on demand computing environment. The blueprint also describes an initial set of core capabilities that enable autonomic computing and discusses technologies that support these core capabilities. It also presents existing, emerging and potential industry standards that are necessary to enable autonomic computing within an open system architecture for heterogeneous environments.

2. Autonomic computing architecture concepts

The architectural concepts presented in this blueprint define a common approach and terminology for describing autonomic computing systems. The autonomic computing architecture concepts provide a mechanism for discussing, comparing and contrasting the approaches that different vendors use to deliver self-managing capabilities in an IT system.

**Autonomic computing system**

This blueprint organizes an autonomic computing system into the layers and parts shown in Figure 2 (on page 9). These parts are connected using a distributed infrastructure that allows the components to collaborate using standard mechanisms such as Web services. The distributed infrastructure can be thought of as a service bus that integrates the various blueprint components, which include:

- Managed resources
- Touchpoints
- Autonomic managers
- An integrated solution console

The lowest layer contains the system components, or managed resources, that make up the IT infrastructure. These managed resources can be any type of resource (hardware or software) and may have embedded self-managing attributes. The next layer incorporates consistent, standard manageability interfaces for accessing and controlling the managed resources. These standard interfaces are delivered through a touchpoint. Layers three and four automate some portion of the IT process using an autonomic manager. A particular resource may have one or more touchpoint autonomic managers, each implementing a relevant control loop. Layer 3 in Figure 2 illustrates this
**Highlights**

**Integrated Solutions Console** - A user interface for one or more administrative tasks. For example, the IBM Integrated Solutions Console integrates the administrative tasks for multiple products and solutions into a single console.

**Orchestrating Autonomic Managers** - An autonomic manager that works with other autonomic managers to provide coordination functions.

---

**Integrated Solutions Console**

Integrated Solutions Console - A technology that provides a common, consistent user interface, based on industry standards and component reuse, and can host common system administrative functions. The IBM Integrated Solutions Console is a core technology of the IBM Autonomic Computing initiative that uses a portal-based interface to provide these common system administrative functions for IBM server, software or storage products.

**Orchestrating Autonomic Managers**

Orchestrating within a Discipline

Orchestrating across disciplines

**Touchpoint Autonomic Managers**

**Managed Resources**

Servers

Storage

Network

Database/Middleware

Application

---

**Figure 2. Autonomic computing reference architecture**

by depicting an autonomic manager for each of the four broad categories that were introduced earlier (self-configuring, self-healing, self-optimizing and self-protecting). Layer four contains autonomic managers that orchestrate other autonomic managers. It is these *orchestrating autonomic managers* that deliver the system wide autonomic computing capability by incorporating control loops that have the broadest view of the overall IT infrastructure. The top layer provides a common system management interface for the IT professional through an integrated solutions console.

The remainder of this chapter describes the function of each layer of the architecture and mentions some core capabilities that are needed for system wide autonomic behavior in the context of the architecture elements.
**Highlights**

*Event - Any significant change in the state of a system resource, network resource or network application. An event can be generated for a problem, for the resolution of a problem or for the successful completion of a task.*

*Sensor - An interface that exposes information about the state and state transitions of a managed resource.*

*Effector - An interface that enables state changes for a managed resource.*

**Managed resources**

A *managed resource* is a hardware or software component that can be managed. A managed resource could be a server, storage unit, database, application server, service, application or other entity. As shown in Figure 2, a managed resource might contain its own embedded self-management control loop with other autonomic managers (described later in this chapter) that might be packaged with a managed resource.

Intelligent control loops can be embedded in the run-time environment of a managed resource. These embedded control loops are one way to offer self-management capability. The details of these embedded control loops may or may not be externally visible. The control loop might be deeply embedded in a resource so that it is not visible through the manageability interface. When any of the details for the control loop are visible, the control loop is configured through the manageability interface (described in Chapter 3) that is provided for that resource (for example, a disk drive).

**Touchpoints**

A *touchpoint* is an autonomic computing system building block that implements sensor and effector behavior for one or more of a managed resource’s manageability mechanisms. It also provides a standard manageability interface. Deployed managed resources are accessed and controlled through these manageability interfaces. Manageability interfaces employ mechanisms such as log files, events, commands, application programming interfaces (APIs) and configuration files. These mechanisms provide various ways to gather details about and change the behavior of the managed resources. In the context of this blueprint, the mechanisms used to gather details are aggregated into a sensor for the managed resource and the mechanisms used to change the behavior of the managed resources are aggregated into an effector for the resource.

**Touchpoint autonomic managers**

Autonomic managers implement intelligent control loops that automate combinations of the tasks found in IT processes. *Touchpoint autonomic managers* are those that work directly with the managed resources through their touchpoints. These autonomic managers can perform various self-management tasks, so they embody different intelligent control loops. Some examples of such control loops, using the four self-managing categories introduced earlier in this paper, include:
• Performing a self-configuring task such as installing software when it detects that some prerequisite software is missing
• Performing a self-healing task such as correcting a configured path so installed software can be correctly located
• Performing a self-optimizing task such as adjusting the current capacity when it observes an increase or decrease in workload
• Performing a self-protecting task such as taking resources offline if it detects an intrusion attempt

Most autonomic managers use policies that provide the goals or objectives for the intelligent control loops. Touchpoint autonomic managers use these policies to determine what actions should be taken for the managed resources that they manage.

A touchpoint autonomic manager can manage one or more managed resources directly, using the managed resource’s touchpoint or touchpoints. Figure 3 illustrates four typical arrangements. The primary differences among these arrangements are the type and number of managed resources that are within the autonomic manager’s scope of control. The four typical arrangements are:

• A single resource scope is the most fundamental because an autonomic manager implements a control loop that accesses and controls a single managed resource, such as a network router, a server, a storage device, an application, a middleware platform or a personal computer.
• A homogeneous group scope aggregates resources of the same type. An example of a homogeneous group is a pool of servers that an autonomic manager can dynamically optimize to meet certain performance and availability thresholds.
• A heterogeneous group scope organizes resources of different types. An example of a heterogeneous group is a combination of heterogeneous devices and servers such as databases, Web servers and storage subsystems that work together to achieve common performance and availability targets.
• A business system scope organizes a collection of heterogeneous resources so an autonomic manager can apply its intelligent control loop to the service that is delivered to the business. Some examples are a customer care system or an electronic auction system. The business system scope requires autonomic managers that can comprehend the optimal state of business processes—based on policies, schedules and service levels—and drive the consequences of process optimization back down to the resource groups (both homogeneous and heterogeneous) and even to individual resources.
These resource scopes define a set of decision-making contexts that are used to classify the purpose and role of a control loop within the autonomic computing architecture.

The touchpoint autonomic managers shown previously in Figure 2 are each dedicated to a particular resource or a particular collection of resources. Touchpoint autonomic managers also expose a sensor and an effector, just like the managed resources in Figure 3 (above), do. As a result, orchestrating autonomic managers (described next) can interact with touchpoint autonomic managers by using the same style of standard interface that touchpoint autonomic managers use to interact with managed resources.

**Orchestrating autonomic managers**

A single touchpoint autonomic manager acting in isolation can achieve autonomic behavior only for the resources that it manages. The self-managing capabilities delivered by touchpoint autonomic managers need to be coordinated to deliver system wide autonomic computing behavior. **Orchestrating autonomic managers** provide this coordination function. There are two common configurations:
• Orchestrating within a discipline: An orchestrating autonomic manager coordinates multiple touchpoint autonomic managers of the same type (one of self-configuring, self-healing, self-optimizing or self-protecting).
• Orchestrating across disciplines: An orchestrating autonomic manager coordinates touchpoint autonomic managers that are a mixture of self-configuring, self-healing, self-optimizing and self-protecting.

An example of an orchestrating autonomic manager is a workload manager. An autonomic management system for workload might include self-optimizing touchpoint autonomic managers for particular resources, as well as orchestrating autonomic managers that manage pools of resources. A touchpoint autonomic manager can optimize the utilization of a particular resource based on application priorities. Orchestrating autonomic managers can optimize resource utilization across a pool of resources, based on transaction measurements and policies. The philosophy behind workload management is one of policy-based, goal-oriented management.

Tuning servers individually using only touchpoint autonomic managers cannot ensure the overall performance of applications that span a mix of platforms. Systems that appear to be functioning well on their own may not, in fact, be contributing to optimal system wide end-to-end processing.

**Integrated solutions console**

Autonomic systems require common console technology to create a consistent human-facing interface for the autonomic managers of IT infrastructure components. As indicated earlier, autonomic capabilities in computer systems make decisions based on tasks that IT professionals choose to delegate to the technology, according to policies. In some cases, an administrator might choose for certain tasks to involve human intervention, and the human interaction with the system can be enhanced using common console technology. The common console capability provides a framework for reuse and consistent presentation for other autonomic technologies. The primary goal of a common console is to provide a single platform that can host all the administrative console functions in server, software and storage products to allow users to manage solutions rather than managing individual components or products. Administrative console functions range from setup and configuration to solution run-time monitoring and control.

The customer value of an integrated solutions console includes reduced cost of ownership—attributable to more efficient administration—and shorter learning curves as new products and solutions are added to the autonomic system environment. The shorter learning curve is achieved by using standards and a
Web-based presentation style. By delivering a consistent presentation format and behavior for administrative functions across diverse products, the common console creates a familiar user interface, reducing the need for staff to learn a different interface each time a new product is introduced.

Because the common console architecture is based on standards (such as standard Java™ APIs and extensions including JSR168, JSR127 and others), it can be extended to offer new management functions or to enable the development of new components for products in an autonomic system.

A common console instance consists of a framework and a set of console-specific components provided by products. Administrative activities are executed as portlets. Consistency of presentation and behavior is essential to improving administrative efficiency, and requires ongoing effort and cooperation among many product communities.

**Autonomic computing blueprint illustration**

Figure 4 (below) shows an example of a simple IT system that includes two business applications: a customer order application and a vendor relationship application. Each of these applications depends on a set of IT resources—databases and servers—to deliver its functionality. Some of these resources—Server B and Company Database—are shared between the applications, which are managed separately.

![Figure 4. Example system with shared resource management](image-url)
The sample system shown in Figure 4 is used to illustrate the key principles of this autonomic computing blueprint. Although simple, this example captures many of the patterns that are found in a typical system and illustrates some of the challenges associated with building an autonomic computing system. These patterns are:

- An IT system is composed of myriad managed resource types. In this example, there are four databases, three servers and two business applications.
- Resources can be organized in topology to gain some efficiency. In the example, multiple resources are placed on the same server to increase the probability that the servers are used as efficiently as possible.
- Resources are shared between applications to enable application integration. In the example, the company database provides common company details for both the customer order and the vendor relationship applications.

At least four groupings, or resource scopes (decision-making contexts), exist in this example. Each application (customer order and vendor relationship) has a business system scope, focused on the business logic that it implements. A homogeneous resource scope is used when managing the common database issues and a second homogeneous resource scope is used when managing common server issues.

We now apply the autonomic computing blueprint to this example. This illustration of the blueprint uses eleven touchpoint autonomic managers and two orchestrating autonomic managers, as illustrated in Figure 5.

The classifications for these thirteen autonomic managers are:

1. Four instances of a touchpoint autonomic manager for a database managed resource.
2. Three instances of a touchpoint autonomic manager for a server managed resource.
3. Two instances of a touchpoint autonomic manager for an application managed resource.
4. One touchpoint autonomic manager that manages a pool of databases as a logical managed resource.
5. One touchpoint autonomic manager that manages a pool of servers as a logical managed resource.
6. One orchestrating autonomic manager for the customer order application. To accomplish its business goals, this autonomic manager manages the touchpoint autonomic managers that are assigned to the IT resources.
(7) One orchestrating autonomic manager for the vendor relationship application. To accomplish its business goals, this autonomic manager manages the touchpoint autonomic managers that are assigned to the IT resources.

Figure 5 illustrates how these thirteen autonomic managers are arranged.

Because the decision-making contexts for these autonomic managers are inter-dependent, the autonomic managers for all the contexts must cooperate.

This coordination among autonomic managers is accomplished by using a distributed infrastructure that makes it possible to identify situations in which there are multiple managers. It also enables autonomic managers to negotiate resolutions for resource scope conflicts, based on a system wide business and resource optimization policy. This system wide collaboration enables a business to realize the benefits of automation across its entire IT infrastructure and set of business processes.
This example illustrates the principles that are employed in a distributed infrastructure. Using sensor and effector interfaces, the various autonomic managers can collaborate in the system wide management, with each manager being responsible for a particular resource scope and playing some role in the management of the overall system.

3. Autonomic computing architecture details

This chapter provides additional details about the architecture concepts introduced in the previous chapter and includes examples of autonomic computing core capabilities associated with these architectural elements.

---

Core Capabilities Example

Throughout this chapter, the examples of workload policies and transaction measurements will illustrate how autonomic managers and managed resources might operate in the context of these two core capabilities. These examples are used again later to illustrate how the various parts of an autonomic manager and a managed resource relate to these core capabilities.

A workload policy contains simple definitions of classes of service—broad categories of “work”—and an associated performance goal for each class of service.

The goals are stated in terms such as “need to complete 90 percent of the transactions in less than one second,” or “an average response time of two seconds.” Transaction measurements are used to determine the extent to which these goals are being met.

In addition to a goal declaration, each class of service has an associated business importance level that indicates how important the achievement of the goal is to the business that owns the computing resources.

The overall policy then becomes quite simple: satisfy the goals of the most important workloads based on the transaction measurements, and then worry about the rest.
Autonomic manager

The autonomic manager is a component that implements the control loop (described earlier in the Autonomic computing section of the Introduction). For a system component to be self-managing, it must have an automated method to collect the details it needs from the system; to analyze those details to determine if something needs to change; to create a plan, or sequence of actions, that specifies the necessary changes; and to perform those actions. When these functions can be automated, an intelligent control loop is formed.

As shown in Figure 6, the architecture dissects the loop into four parts that share knowledge:

- The **monitor function** provides the mechanisms that collect, aggregate, filter and report details (such as metrics and topologies) collected from a managed resource.
- The **analyze function** provides the mechanisms that correlate and model complex situations (for example, time-series forecasting and queuing models). These mechanisms allow the autonomic manager to learn about the IT environment and help predict future situations.
- The **plan function** provides the mechanisms that construct the actions needed to achieve goals and objectives. The planning mechanism uses policy information to guide its work.
- The **execute function** provides the mechanisms that control the execution of a plan with considerations for dynamic updates.

These four parts work together to provide the control loop functionality. Figure 6 shows a structural arrangement of the parts rather than a control flow. The four parts communicate and collaborate with one another and exchange appropriate knowledge and data, as shown in Figure 6.
As illustrated in Figure 6 (above), autonomic managers, in the same manner as touchpoints, provide sensor and effector interfaces for other autonomic managers and other components in the distributed infrastructure to use. Using sensor and effector interfaces for the distributed infrastructure components enables these components to be composed together in a manner that is transparent to the managed resources. For example, an orchestrating autonomic manager can use the sensor and effector interfaces of touchpoint autonomic managers to accomplish its management functions, as illustrated previously in Figure 5.

Even though an autonomic manager is capable of automating the monitor, analyze, plan and execute parts of the loop, IT professionals can configure the autonomic manager to perform only part of its automated function. In Figure 6, four profiles (monitoring, analyzing, planning and executing) are shown. An administrator might configure this autonomic manager to perform only the monitoring function. As a result, the autonomic manager would surface notifications to a common console for the situations that it recognizes, rather than automating the analysis, planning and execution functions associated with those actions. Other configurations could allow additional parts of the control loop to be automated. An evolutionary process for increasing autonomic function is described in Chapter 4.
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 6 and described in the following sections.

Monitor

The monitor function collects the details from the managed resources, via touchpoints, and organizes them into symptoms that need to 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 the log files for multiple resources to determine a symptom that relates to that particular combination of log entries. Logically, this symptom is passed to the analyze function.

Core Capabilities Example for Monitor

Following the example of transaction measurements and workload policies, the monitor function collects details about initiating and completing transactions. It then calculates the response time for the transaction measurements. If the response time exceeds a specified threshold, the monitor component could produce a “missed response time” symptom.

These transaction measurements are passed to the autonomic manager through the retrieve-state and receive-notification interaction styles associated with a sensor (detailed later). The monitored data could then be used elsewhere in the autonomic manager (for example, during the analyze function to generate a change request or the execute function to perform system self-optimization). Selected monitored data also could be presented to an IT professional on an integrated solutions console.
Autonomic managers must collect and process large amounts of data from the touchpoint 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 such as time-series forecasting and queuing models. These mechanisms allow the autonomic manager to learn about the IT environment and help predict future behavior.

**Core Capabilities Example for Analyze**

The analyze function uses the policies set forth by IT staff to determine the need for changes in the system. These policies specify the criteria that an autonomic manager uses to realize a definite goal or to accomplish a course of action.

Following the core capabilities example used throughout this chapter, an autonomic manager involved in workload management uses the specified workload policies to guide its analysis and decisions. These workload policies might include quality-of-service goals for accomplishing the workload.

Using the symptom data delivered by the monitor function, the analyze function can understand how the involved systems commit their resources to execute the workload and how changes in allocation affect performance over time. It then uses the transaction measurements (from the monitor component) and performance goals (from the workload policies) to generate a change request to alter resource allocations (for example, add or delete resources) to optimize performance across these multiple systems.
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 on page 24.

If changes are required, the analyze function passes a 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 passes the appropriate change plan, which represents a desired set of changes for the managed resource, to the execute function.

**Core Capabilities Example for Plan**

Following the core capabilities example used throughout this chapter, the autonomic manager plan function uses the change request created by the analyze function to generate or select a change plan for improving workload management. For example, the plan function might produce a workflow that will add or delete specific resources or redistribute portions of the workload to different resources.

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

(detailed on page 25) 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).

Core Capabilities Example for Execute
Following the core capabilities example used throughout this chapter, the autonomic manager execute function involved in workload management uses the change plan delivered by the plan function to make the necessary modifications to the managed resource. These modifications are made using the touchpoint effector interface and could include creating, deleting, starting or stopping specific resources instances.

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.

The knowledge used by an autonomic manager is obtained in one of three ways:

(1) The knowledge is passed to the autonomic manager through its effector interface. An autonomic manager might obtain policy knowledge in this manner. A policy consists of a set of behavioral constraints or preferences that influence the decisions made by an autonomic manager.

(2) The knowledge is retrieved from an external information service. An autonomic manager might obtain resource-specific historical knowledge in this manner. A log file may contain a detailed history in the form of entries that signify events that have occurred in a component or system.

(3) The autonomic manager itself creates the knowledge. The knowledge used by a particular autonomic manager could be created by the monitor part, based on the information collected through sensors. The monitor part might create knowledge based on recent activities by logging the notifications that it receives from a managed resource. The execute part of an autonomic manager might update the knowledge to indicate the actions that were taken as a result of the analysis and planning (based on the monitored data). The execute part would then indicate how those actions affected the managed resource (based on subsequent monitored data obtained from the managed resource after the actions were carried out).
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 an autonomic system. Each knowledge type must be expressed using common syntax and semantics so the knowledge can be shared.

<table>
<thead>
<tr>
<th>Knowledge Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution Topology Knowledge</td>
<td>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.</td>
</tr>
<tr>
<td>Policy Knowledge</td>
<td>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.</td>
</tr>
<tr>
<td>Problem Determination Knowledge</td>
<td>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.</td>
</tr>
</tbody>
</table>

Managed resource
The managed resource, depicted in Figure 7 (on page 26), is a controlled system component. It can be a single resource (for example, a server, database server or router) or a collection of resources (for example, a pool of servers, a cluster or a business application).
An autonomic manager communicates with a managed resource through the manageability interface, described next. A touchpoint is the implementation of the manageability interface by a specific managed resource. For example, a database server might implement a touchpoint for communicating with an autonomic manager.

Manageability Interface

The manageability interface for controlling a managed resource is organized into its sensor and effector. A touchpoint implements the sensor and effector behavior for a specific managed resource by mapping the standard sensor and effector interfaces to one or more of the managed resource’s manageability interface mechanisms. The touchpoint reduces complexity by offering a standard interface to autonomic managers, rather than the numerous, diverse manageability interface mechanisms associated with the many types of managed resources.

Sensor

A sensor consists of one or both of the following:

- A set of “get” operations that retrieve information about the current state of a managed resource
- A set of management events (unsolicited, asynchronous messages or notifications) that occur when the managed resource undergoes significant state changes
These two parts of a sensor interface are referred to as interaction styles. The “get” operations use the retrieve-state interaction style; events use the receive-notification interaction style.

**Effector**

An effector consists of one or both of the following:

- A collection of “set” operations that allow the state of the managed resource to be changed in some important way
- A collection of operations that the managed resource can use to make requests

The “set” operations use the perform-operation interaction style; requests use the call-out request interaction style to allow the managed resource to consult with some external entity.

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 management topics.

A management topic refers to a logical collection of managed resource state information and operations. Some examples of management topics are:

- Identification: state information and operations used to identify an instance of a managed resource
- Metrics: state information and operations for measurements of a managed resource, such as throughput, utilization and so on
- Configuration: state information and operations for the configurable attributes of a managed resource

For each management topic, the client of the manageability interface must be able to obtain and control state data through the manageability interface, including:

- Meta details (for example, properties that are used to identify a managed resource, or information that specifies which resources can be hosted by the managed resource)
- Sensor interactions, including mechanisms for retrieving the current property values (e.g., metrics, configuration) and available notifications (what types of events and situations the managed resource can generate)
- Effector interactions, including operations to change the state (which effector
**Autonomic Computing Maturity Index - A graduated scale that expresses the level of maturity of autonomic computing, where level 1 is basic (completely manual), level 2 is managed, level 3 is predictive, level 4 is adaptive, and level 5 is fully autonomic.**

4. Evolving maturity and sophistication

Incorporating self-managing capabilities into an IT environment is an evolutionary process. It is ultimately implemented by each organization through the adoption of autonomic computing technologies, supporting processes and skills. Throughout this evolution, the computer industry will further develop self-management technologies to help continue to improve staff productivity, increase business resiliency and, ultimately, reduce operating costs.

This evolution toward more highly autonomic capabilities can be described with five autonomic maturity levels, illustrated in Figure 8 (below).

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Basic</th>
<th>Managed</th>
<th>Predictive</th>
<th>Adaptive</th>
<th>Autonomic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rely on system reports, product documentation, and manual actions to configure, optimize, heal and protect individual IT components</td>
<td>Management software in place to provide consolidation, facilitation and automation of IT tasks</td>
<td>Most analysis and associated actions determined and implemented by IT staff</td>
<td>IT components, individually and collectively, able to monitor, analyze and take action with minimal human intervention</td>
<td>IT components collectively and automatically managed by business rules and policies established in the system</td>
</tr>
<tr>
<td></td>
<td>Requires extensive, highly skilled IT staff</td>
<td>Increased ability to operate complex systems</td>
<td>Reduced dependency on deep skills</td>
<td>Business policy drives IT management</td>
<td></td>
</tr>
<tr>
<td>Incremental Benefits</td>
<td></td>
<td></td>
<td>Improved decision making</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8. The autonomic computing maturity index**
• At the basic level, IT professionals manage each resource independently and set it up, monitor it and eventually replace it.
• At the managed level, systems management technologies can be used to collect details from disparate systems into fewer consoles, helping to reduce the time it takes for the administrator to collect and synthesize information as the IT environment becomes more complex.
• At the predictive level, new technologies are introduced to provide correlation among several managed resources. The management functions can begin to recognize patterns, predict the optimal configuration and offer advice about what course of action the administrator should take. As these technologies improve and as people become more comfortable with the advice and predictive power of these systems, the technologies can progress to the adaptive level.
• At the adaptive level, the IT environment can automatically take actions based on the available information and the knowledge about what is happening in the environment.
• At the autonomic level, business policies and objectives govern the IT infrastructure operation. Users interact with the autonomic technology tools to monitor business processes, alter the objectives or both.

How the architecture supports autonomic computing evolution
The preceding discussion about autonomic maturity levels demonstrates that self-managing capabilities are not incorporated all at once. Rather, they constitute a concept that permeates all aspects of a system. Figure 9 (on page 29) reinforces this observation by showing one possible relationship among the maturity levels, the various resource scopes and the parts of the autonomic manager. This mapping results in two important observations:

The first relationship illustrated in Figure 9 is the relationship between the maturity levels and the parts of an autonomic manager. As described earlier, different parts of a control loop can be automated. As the level of autonomic maturity increases, more functions of the autonomic manager are automated. In Figure 9, this is illustrated by the autonomic manager functions highlighted in red. These diagrams show possible ways in which management functions might be automated at each level of autonomic maturity. In this example, the monitor and execute functions of the autonomic manager are automated at the basic and managed levels, but IT professionals are responsible for performing the analyze and plan functions. Additional parts of the control loop are automated as the organization progresses to higher levels of autonomic maturity.
The second relationship illustrated in Figure 9 is the relationship between the degree of automation at each maturity level and the resource scopes. In addition to automating more functions of an autonomic manager as the maturity level increases, the automated functions also are applied to broader resource scopes. In Figure 9, the pyramid illustrates these resource scopes, and the autonomic manager diagrams show how the levels of automation might vary across resource scopes at a particular maturity level. In this example, all the functions of an autonomic manager are automated within the single resource scope (in which we would expect to see touchpoint autonomic managers) at the predictive level, but IT professionals are responsible for performing the plan function within the resource group scope and the analysis and plan functions within the business system scope. In addition to increasing levels of automation, the automation is applied across broader scopes as the organization progresses to higher levels of autonomic maturity.

5. Standards for autonomic computing

The fundamental nature of autonomic computing systems precludes any single company from delivering an entire autonomic solution. Businesses have heterogeneous IT infrastructures and must deal with heterogeneous environments outside of the enterprise. A proprietary implementation would be like a heart that maintains a regular steady heartbeat but cannot adjust to the needs of the body when under stress. Autonomic computing systems require autonomic managers to be deployed across the IT infrastructure, managing various resources (including other autonomic managers) from a diverse range of suppliers. Therefore, these systems must be based on open industry standards. This blueprint identifies relevant existing computing industry standards. New open standards will be developed and shared in the industry to define the mechanisms for interoperating in a heterogeneous system environment.

Examples of existing and emerging standards relevant to autonomic computing are described in Table 2 (on page 30).
<table>
<thead>
<tr>
<th>Related to Autonomic Managers</th>
<th>Related to Touchpoints</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Distributed Management Task Force (DMTF)</strong></td>
<td></td>
</tr>
<tr>
<td>Common Information Model (CIM), Web Services Common Information Model (WS-CIM)</td>
<td>✓</td>
</tr>
<tr>
<td>Applications Working Group</td>
<td>✓</td>
</tr>
<tr>
<td>Utility Computing Working Group</td>
<td>✓</td>
</tr>
<tr>
<td>Server Management Working Group</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Internet Engineering Task Force (IETF)</strong></td>
<td></td>
</tr>
<tr>
<td>Policy - Core Information Model (RFC3060)</td>
<td>✓</td>
</tr>
<tr>
<td>Simple Network Management Protocol (SNMP)</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Organization for the Advancement of Structured Information Standards (OASIS)</strong></td>
<td></td>
</tr>
<tr>
<td>Web Services Security (WS-Security)</td>
<td>✓</td>
</tr>
<tr>
<td>Web Services Distributed Management (WS-DM)</td>
<td>✓</td>
</tr>
<tr>
<td>Web Services Resource Framework (WS-RF)</td>
<td>✓</td>
</tr>
<tr>
<td>Web Services Notification (WS-N)</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Java Community Process</strong></td>
<td></td>
</tr>
<tr>
<td>Java Management Extensions (JSR3, JMX)</td>
<td>✓</td>
</tr>
<tr>
<td>Logging API Specification (JSR47)</td>
<td>✓</td>
</tr>
<tr>
<td>Java Agent Services (JSR87)</td>
<td>✓</td>
</tr>
<tr>
<td>Portlet Specification (JSR168)</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Storage Networking Industry Association (SNIA)</strong></td>
<td></td>
</tr>
<tr>
<td>Storage Management Initiative Specification (SMI-S)</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Global Grid Forum (GGF)</strong></td>
<td></td>
</tr>
<tr>
<td>Open Grid Services Architecture (OGSA)</td>
<td>✓</td>
</tr>
<tr>
<td>Open Grid Services Infrastructure (OGSI)</td>
<td>✓</td>
</tr>
<tr>
<td>Open Grid Services Common Management Model (CMM-Working Group)</td>
<td>✓</td>
</tr>
<tr>
<td>Grid Resource Allocation Agreement Protocol (GRAAP-Working Group)</td>
<td>✓</td>
</tr>
<tr>
<td><strong>The Open Group</strong></td>
<td></td>
</tr>
<tr>
<td>Application Response Measurement (ARM)</td>
<td>✓</td>
</tr>
<tr>
<td><strong>World Wide Web Consortium (W3C)</strong></td>
<td></td>
</tr>
<tr>
<td>Solution Install Schema</td>
<td>✓</td>
</tr>
<tr>
<td><strong>New standards to be developed</strong></td>
<td>✓</td>
</tr>
</tbody>
</table>

*Table 2. Examples of standards related to autonomic computing*
Open standards developed in the appropriate standards bodies are vital to enable the evolution of autonomic computing.

This architecture does not prescribe a particular management protocol or instrumentation technology because the architecture needs to work with the various computing technologies and standards that exist in the industry today—SNMP, Java™ Management Extensions (JMX), Distributed Management Task Force, Inc. (DMTF)—as well as future technologies.

Given the diversity of these management technologies that already exist in the IT industry, this architecture endorses Web services techniques for sensors and effectors. These techniques encourage implementers to leverage existing approaches and support multiple binding and marshalling techniques.

6. Summary

Autonomic computing is about shifting the burden of managing systems from people to technologies. When the self-management capabilities delivered by IBM and other vendors can collaborate, the elements of a complex IT system can work together and manage themselves based on a shared view of system wide policy and objectives.

This paper has presented a high-level architectural blueprint to assist in delivering autonomic computing in phases. The architecture reinforces that self-management uses intelligent control loop implementations to monitor, analyze, plan and execute, leveraging knowledge of the environment. These control loops can be embedded in resource run-time environments or delivered in management tools. The control loops collaborate using a distributed infrastructure among autonomic managers and touchpoints.

Autonomic managers communicate with managed resources through the manageability interface, in the form of a touchpoint, using sensor and effector interfaces. A sensor interface exhibits two interaction styles, the retrieve-state interaction style (used to query information from a managed resource) and the receive-notification interaction style (used to send asynchronous event information from a managed resource). The effector interface exhibits two interaction styles, the perform-operation interaction style (used to set state data in the managed resource) and the call-out request interaction style (used by a managed resource to obtain services from some other external entity in the system).

---

**Highlights**

**Web service** - A self-contained, modular application that can be described, published, located and invoked over a network (generally the Internet). Web services go beyond software components, because they can describe their own functionality, as well as look for and dynamically interact with other Web services. Web services use open protocols and standards, such as HTTP, SOAP and XML. Web services provide a means for different organizations to connect their applications with one another to conduct dynamic e-business across a network, regardless of their application, design or run-time environment.
A number of core capabilities inherent in an autonomic computing system have been identified and this paper has described how the autonomic computing architecture elements support and enable core capabilities, using two examples (transaction measurements and workload policies).

The journey to a fully autonomic IT infrastructure is an evolution. The aspects of the architecture that need to be addressed at the five management maturity levels illustrate the stages of this evolution. This model is then applied to the IT infrastructure using resource scopes (single resource, homogeneous and heterogeneous groups, and business system resource scopes).

Businesses—small, medium and large—want and need to reduce their IT costs, simplify the management of complex IT resources, realize a faster return on their IT investments, and ensure the highest possible levels of system availability, performance, security and asset utilization. Autonomic computing addresses these issues—not just through new technology but also through a fundamental, evolutionary shift in the way that IT systems are managed. Moreover, autonomic computing will free IT staffs from detailed mundane tasks, allowing them to focus on managing business processes. True autonomic computing will be accomplished through a combination of process changes, skills evolution, new technologies, architecture and open industry standards.

For more information
Please contact your IBM marketing representative or an IBM Business Partner, or call 1-800-IBM-CALL within the U.S.

Visit us at ibm.com/autonomic