Autonomic Computing
White Paper

An architectural blueprint for autonomic computing.

June 2005
Third Edition

Contents

1. Introduction

In an on demand business, information technology (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, 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 system 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.

Previous versions of this Blueprint have described the overall Autonomic Computing architecture. In this latest version, we describe not only the latest advances in the architecture but also offer more detailed discussions of several important topics, including:

- Architectural building blocks
- The Autonomic Computing Adoption Model
- The role of the human in autonomic systems, including delegation of tasks
- New developments in standardization

Highlights

Autonomic - Pertaining to an on demand operating environment that responds automatically to problems, security threats, and system failures.

Autonomic computing - A computing environment with the ability to manage itself and dynamically adapt to change in accordance with business policies and objectives. Self-managing environments can perform such activities based on situations they observe or sense in the IT environment rather than requiring IT professionals to initiate the task. These environments are self-configuring, self-healing, self-optimizing, and self-protecting.

Quality of Service - A measure of system performance and system availability.

Highlights

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

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

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

Autonomic Computing

Autonomic Computing helps to address 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, self-managing autonomic 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, self-managing autonomic capabilities in computer systems perform 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. This topic is explored in more detail later.

Self-managing 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).

Self-management attributes of system components

In a self-managing autonomic 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

Highlights

Self-Optimize - To tune resources and balance workloads to maximize the use of information technology resources.

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-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-optimizing*–Can anticipate, detect, identify and protect against threats from anywhere 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.

Highlights

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

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. Systems management software then can orchestrate the systemwide actions performed by these embedded control loops.

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 (ITIL, from the Office of Government Commerce in the United Kingdom) and the IBM IT Process Model (developed by IBM Global Services). Figure 1 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.

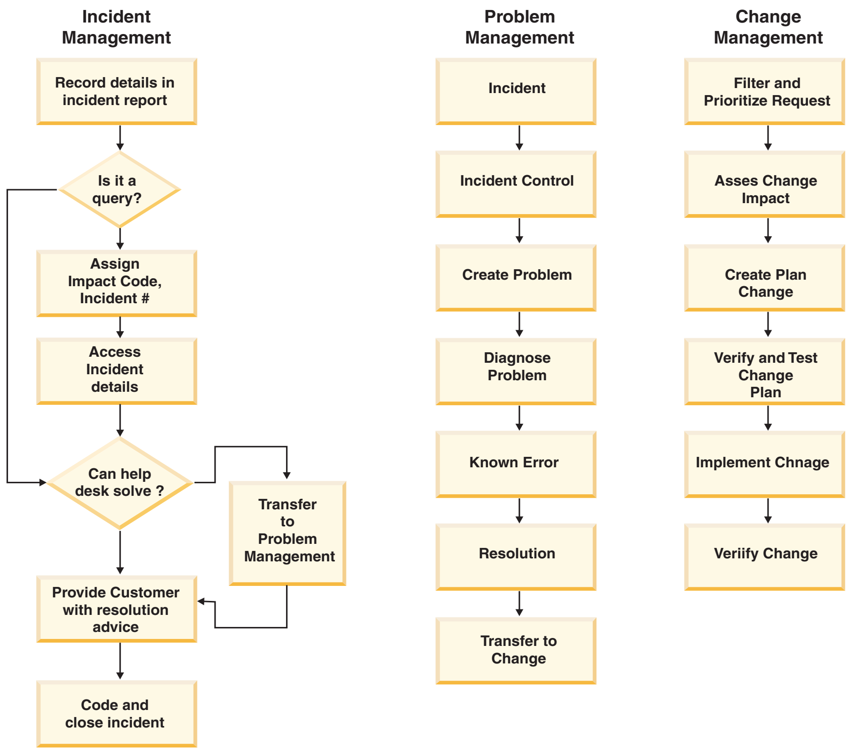

Figure 1. Typical IT processes

Highlights

Manual Manager - An implementation of the user interface that enables a human to perform some management function manually.

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.

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 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.
(3) The autonomic manager must possess sufficient knowledge to take on the delegated task that is to be automated.

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 capabilities typically are delivered as management tools or products.

Automating these tasks involves an IT professional delegating the automatable task to the system. The manual manager building block (detailed later) is the architectural representation of the human activity and typically involves a human using a management console. As the IT professional observes situations within the IT infrastructure, he or she may take certain actions to affect the behavior of the system.

Over time, as autonomic monitoring of the IT system is introduced, an IT professional might find that he or she is observing the same conditions and performing the same actions repeatedly. Such cases are good candidates for automation, and the IT professional might choose to delegate these tasks to an autonomic manager that has the requisite knowledge and capabilities to recognize the situations and perform the appropriate actions.

For example, consider the "implement change" task in the change management process in Figure 1. One type of change that might be affected on a system is to provision new resources. The IT professional might choose to delegate some such tasks to a provisioning system. This system could automate the provisioning of servers and network resources, as well as the distribution and installation of software, and hence could automate the "implement change" task in the change management process.

For an IT professional to be willing to delegate management tasks to the system, he or she must have a high degree of trust in the autonomic management functions. As described in Chapter 4, moving toward higher degrees of autonomic maturity is an evolutionary process. One phase of this process involves management functions that can monitor the IT system for situations of interest, perform analysis of those situations, generate recommended changes to the IT system and present those changes to a manual manager (IT professional) for evaluation. This phase is an important one, as it enables the IT professional to build trust in the autonomic management functions – that is, if the autonomic manager consistently recommends actions that the IT professional routinely performs, then the IT professional is likely to become willing to automate those actions by delegating the corresponding tasks to the autonomic manager. This, in turn, enables continued evolution to a "closed loop" degree of autonomic maturity as described in Chapter 4.

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. Self-managing 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.

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.

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 infrastructure 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. Self-managing autonomic computing can result in a significant improvement in system management efficiency. However, this is possible only when the disparate technologies that manage the IT environment work together to deliver performance results systemwide. For this to happen in a multi-vendor infrastructure, IBM and other vendors must agree on a standards-based approach for autonomic systems.

This architectural blueprint for autonomic computing is an overview of the fundamental concepts, constructs and behaviors for building self-managing autonomic capability into an on demand computing environment. The blueprint also describes the architectural building blocks used to construct such autonomic capabilities and a model for adoption of those capabilities in businesses. It also presents recent developments in industry standards initiatives that 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 self-managing 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.

Highlights

***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** - 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.*

***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.*

Autonomic computing system

This blueprint organizes an autonomic computing system into the layers and parts shown in Figure 2. These parts are connected using enterprise service bus patterns that allow the components to collaborate using standard mechanisms such as Web services. The enterprise service bus integrates the various blueprint building blocks, which include:

- Touchpoints for managed resources
- Knowledge sources
- Autonomic managers
- Manual managers

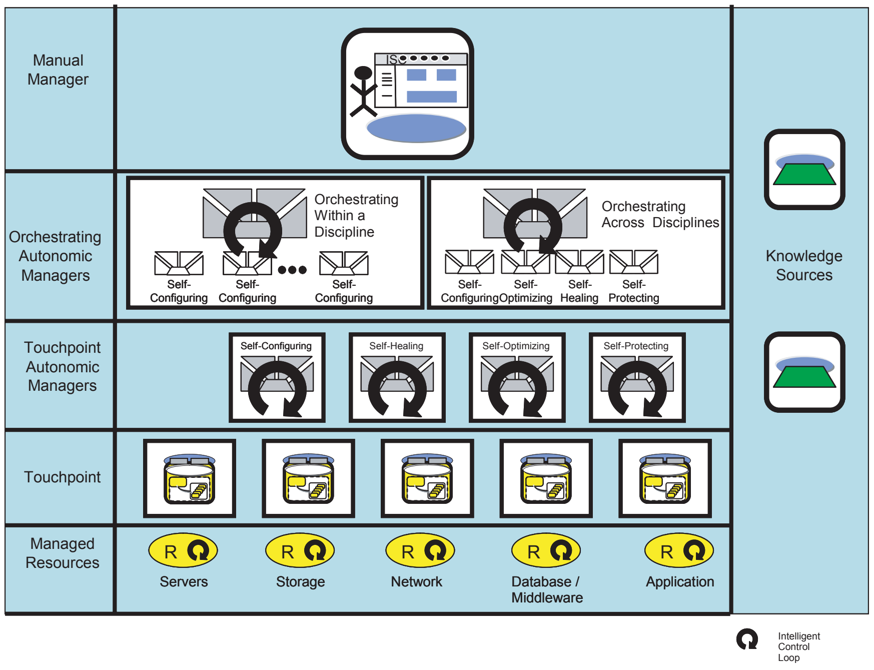

Figure 2. Autonomic computing reference architecture

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.

Highlights

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

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 Manager - An autonomic manager that works with other autonomic managers to provide coordination functions.

Manageability Interface - A service of the managed resource that includes the sensor and effector used by an autonomic manager. The autonomic manager uses the manageability interface to monitor and control the managed resource.

A particular resource may have one or more touchpoint autonomic managers, each implementing a relevant control loop. Layer 3 in Figure 2 illustrates this by depicting an autonomic manager for 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 systemwide autonomic capability by incorporating control loops that have the broadest view of the overall IT infrastructure. The top layer illustrates a manual manager that provides a common system management interface for the IT professional using an integrated solutions console. The various manual and autonomic manager layers can obtain and share knowledge via *knowledge sources*.

The remainder of this chapter describes the function of each of these parts of an autonomic computing system. Chapter 3 then details these architectural concepts.

Managed resource

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, in addition to 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-managing autonomic 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 workload when it observes an increase or decrease in capacity
- Performing a self-protecting task such as taking resources offline if it detects an intrusion attempt

Most autonomic managers use policies (goals or objectives) to govern the behavior of 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.

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.

Enterprise Service Bus - An implementation that assists in integrating other building blocks (for example, autonomic managers and touchpoints) by directing the interactions among these building blocks.

Knowledge Source - An implementation of a registry, dictionary, database or other repository that provides access to knowledge according to the interfaces prescribed by the architecture.

.

- 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.

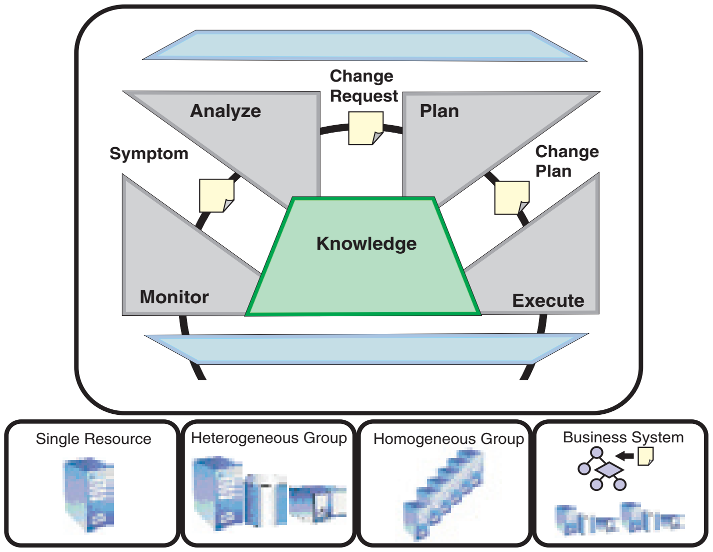

Figure 3. Four common managed resource arrangements

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 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 autonomic capabilities delivered by touchpoint autonomic managers need to be coordinated to deliver systemwide 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 systemwide end-to-end processing.

Manual Managers

A manual manager provides a common system management interface for the IT professional using an integrated solutions console. Self-managing autonomic systems can use 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 perform 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 a common console framework, based on industry standards, that promotes consistent presentation to IT professionals. 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.

The common console architecture is based on standards (such as standard Java APIs and extensions including JSR168, JSR127 and others), so that 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.

3. Autonomic computing architecture details

This chapter provides additional details about the architectural concepts introduced in the previous chapter and details the architectural building blocks.

The five building blocks for an autonomic system are:

- Autonomic manager
- Knowledge source
- Touchpoint
- Manual manager
- Enterprise service bus

These building blocks are the architectural representations of the components in an autonomic system and they work together to provide self-managing capabilities. Each of the five building blocks is detailed next.

Autonomic manager

An autonomic manager is an implementation that automates some management function and externalizes this function according to the behavior defined by management interfaces. 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 4, 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 4 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 4.
As illustrated in Figure 4, autonomic managers, in a manner similar to

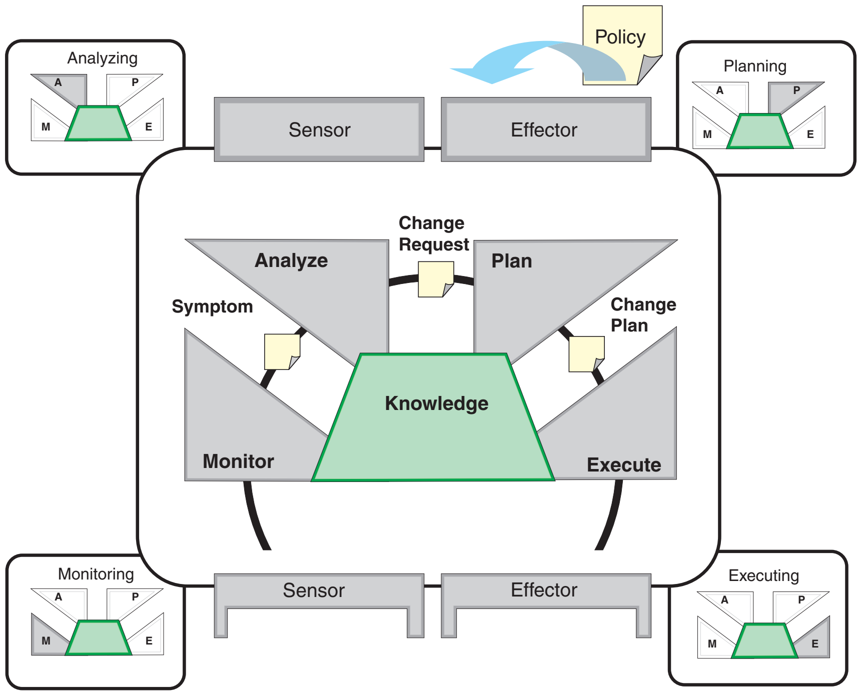

Figure 4. Functional details of the autonomic manager

touchpoints, provide sensor and effector manageability interfaces for other autonomic managers and manual managers to use. Using standard sensor and effector interfaces 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 manageability interfaces of touchpoint autonomic managers to accomplish its management functions (that is, the orchestrating autonomic manager can manage touchpoint autonomic managers), as illustrated previously in Figure 2.

Even though an autonomic manager is capable of automating the monitor, analyze, plan and execute parts of the loop, partial autonomic managers that perform only a subset of the monitor, analyze, plan and execute functions can be developed, and IT professionals can configure an autonomic manager to perform only some of the automated functions it is capable of performing. In Figure 4, 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. Autonomic managers

Highlights

Correlation - The process of analyzing event data to identify patterns, common causes and root causes. Event correlation analyzes the incoming events for known states, using rules and relationships.

that perform only certain parts of the control loop can be composed together to form a complete closed loop. For example, one autonomic manager that performs only the monitor and analyze functions might collaborate with another autonomic manager that performs only the plan and execute functions to realize a complete autonomic control loop. 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 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 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.

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.

Highlights

Monitor Function - The autonomic manager function that collects, aggregates, filters and reports details (such as metrics, topologies) that were collected from managed resources.

Analyze Function -The autonomic manager function that correlates and models complex situations, such as time-series forecasting or queuing models, to understand the current system state.

Plan Function - The autonomic manager function that structures the actions needed to achieve goals and objectives.

Execute Function - The autonomic manager function that changes the behavior of the managed resource using effectors, based on the actions recommended by the plan function.

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 touchpoint 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).

Highlights

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

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.

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

(1) The knowledge is passed to the autonomic manager.
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 knowledge source.
An autonomic manager might obtain symptom definitions or resource-specific historical knowledge in this manner. A knowledge source could store symptoms that could be used by an autonomic manager; a log file may contain a detailed history in the form of entries that signify events that have occurred in a component or system.

Highlights

Installable Unit - An entity that is deployed into an IT system to create new capabilities in that IT system. An installable unit consists of a descriptor and one or more artifacts that need to be installed.

(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). This knowledge is contained within the autonomic manager, as represented by the "knowledge" block in Figure 4. If the knowledge is to be shared with other autonomic managers, it must be placed into a knowledge source.

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.

Highlights

Common Base Event - The standard format and content specification for the structure of events that are sent as the result of a situation and subsequently used by enterprise management and business applications. The common base event includes logging, tracing, management and business events.

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.

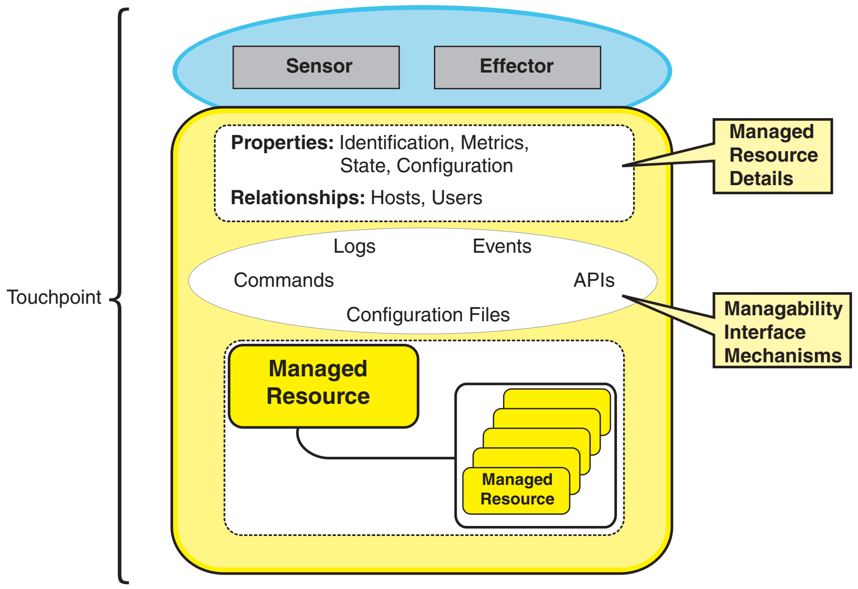

Figure 5. Managed resource touchpad

Manageability Interface

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

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. Some examples of manageability capabilities are:

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

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

- Meta details (for example, to identify properties that are used for configuration of a manageable resource, or information that specifies which resources can be hosted by the manageable resource)
- Sensor interactions, including mechanisms for retrieving the current property values (such as metrics, configuration) and available notifications (what types of events and situations the manageable resource can generate)
- Effector interactions, including operations to change the state (which effector operations and interaction styles the manageable resource supports) and call-outs to request changes to existing state (what types of call-outs the manageable resource can perform)

Manual Manager

A manual manager is an implementation of the user interface that enables an IT professional to perform some management function manually. The manual manager can collaborate with other autonomic managers at the same level or orchestrate autonomic managers and other IT professionals working at "lower" levels.

The manual manager building block is the architectural representation of the human activity and typically involves a human using a management console, as described in Chapter 2. A manual manager can enable an IT professional to delegate management functions to autonomic managers, as described in Chapter 1.

Enterprise Service Bus

An enterprise service bus is an implementation that assists in integrating other building blocks (for example, autonomic managers and touchpoints) by directing the interactions among these building blocks.

The enterprise service bus can be used to “connect” various autonomic computing building blocks. The role that a particular logical instance of the enterprise service bus performs is established by autonomic computing usage patterns such as:

- An enterprise service bus that aggregates multiple manageability mechanisms for a single manageable resource;
- An enterprise service bus that enables an autonomic manager to manage multiple touchpoints;
- An enterprise service bus that enables multiple autonomic managers to manage a single touchpoint; and
- An enterprise service bus that enables multiple autonomic managers to manage multiple touchpoints.

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 self-managing autonomic technologies, supporting processes and skills. Throughout this evolution, the computer industry will further develop self-managing technologies to help continue to improve staff productivity, reduce operating costs and ultimately, increase business resiliency.

This evolution toward more highly autonomic capabilities can be described using the autonomic computing adoption model, discussed next.

Autonomic Computing Adoption Model

Figure 6 depicts the autonomic computing adoption model.

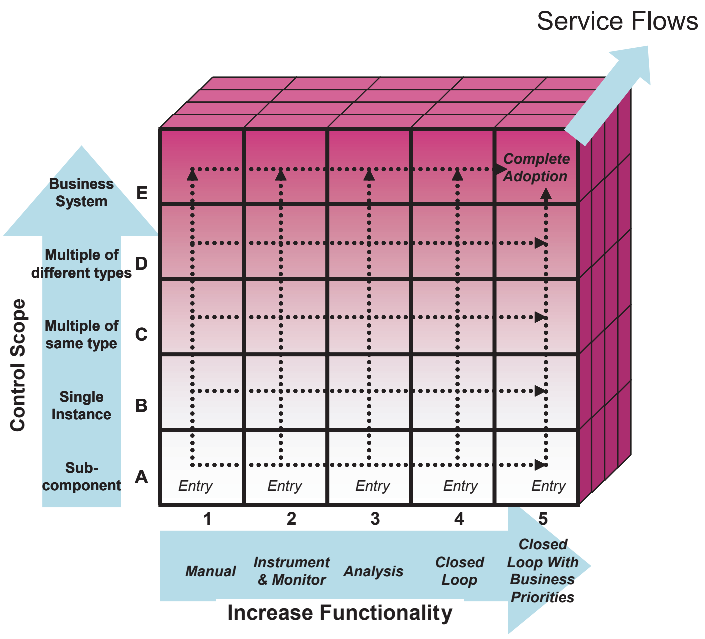

Figure 6. The autonomic computing adoption model

The Autonomic Computing Adoption Model, developed by IBM Global Services, provides a methodology for businesses to calibrate the degree of autonomic capability that their current infrastructure and organization has and to develop action plans to increase the autonomic potential.

The "functionality" dimension, along the x-axis of Figure 6, characterizes the extent of automation of the IT and business processes. Five levels of automation are defined:

- At the manual level, IT professionals perform the management functions.
- At the instrument and monitor level, systems management technologies can be used to collect details from managed resources, helping to reduce the time it takes for the administrator to collect and synthesize information as the IT environment becomes more complex.

- At the analysis 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 closed loop level.
- At the closed loop 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 closed loop with business processes 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.

Notice how five levels, depicted in Figure 6, correspond to the partial autonomic management functions shown in Figure 4.

The "control scope" dimension, along the y-axis of Figure 6, characterizes what is being managed. This dimension also defines five levels of resource management scope:

- At the subcomponent level, portions of resources are managed, such as an operating system on a server or certain applications within an application server.
- At the single instance level, an entire standalone resource is managed, such as a server or complete application server environment.
- At the multiple instances of the same type level, homogeneous resources are managed, typically as a collection, such as a server pool or cluster of application servers.
- At the multiple instances of different types level, heterogeneous are managed as a subsystem, such as a collection of servers, storage units and routers or a collection of application servers, databases and queues.
- At the business system level, a complete set of hardware and software resources that perform business processes is managed from the business process perspective, such as a customer relationship management system or an IT change management system.

The "service flow" dimension, along the z-axis of Figure 6, captures the combination of IT management process activities that are being performed. These service flows do incorporate ITIL processes such as change management, incident management, problem management and so on. Various business and IT processes might demonstrate different maturity levels (in terms of automation and control scope) at the same time, as various tasks and activities within particular service flows are automated.

So autonomic maturity can evolve in three dimensions:

- automating more functions as the maturity level increases
- applying automated functions to broader resource scopes
- automating a range of tasks and activities in various IT management processes

In addition to increasing levels of automation, the automation is applied across broader scopes and within more processes as the organization progresses to higher levels of autonomic maturity. Of course, increasing the autonomic maturity could also involve changes in procedures, skills and organization as more tasks and activities are handled by the technology itself.

The adoption model supports autonomic computing evolution by enabling incremental adoption of additional autonomic capabilities. The adoption model structures a solution space so that a business can produce an incremental action plan to take advantage of offered autonomic capabilities.

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 can not adjust to the needs of the body when under stress. Self-managing 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, followed by a discussion of two recent developments in the area of new standards.

	Related to Autonomic Managers	Related to Touchpoints
Distributed Management Task Force (DMTF)		
Common Information Model (CIM), Web Services Common Information Model (WS-CIM)		✓
Applications Working Group		✓
Utility Computing Working Group		✓
Server Management Working Group		✓
Internet Engineering Task Force (IETF)		
Policy - Core Information Model (RFC3060)	✓	
Simple Network Management Protocol (SNMP)		✓
Organization for the Advancement of Structured Information Standards (OASIS)		
Web Services Security (WS-Security)	✓	✓
Web Services Distributed Management (WS-DM)	✓	✓
Web Services Resource Framework (WS-RF)	✓	✓
Web Services Notification (WS-N)	✓	✓
Java™ Community Process		
Java™ Management Extensions (JSR3, JMX)	✓	
Logging API Specification (JSR47)		✓
Java™ Agent Services (JSR87)		✓
Portlet Specification (JSR168)	✓	
Storage Networking Industry Association (SNIA)		
Storage Management Initiative Specification (SMI-S)		✓
Global Grid Forum (GGF)		
Open Grid Services Architecture (OGSA)	✓	
Open Grid Services Infrastructure (OGSI)	✓	
Open Grid Services Common Management Model (CMM-Working Group	✓	✓
Grid Resource Allocation Agreement Protocol (GRAAP-Working Group)	✓	
The Open Group		
Application Response Measurement (ARM)		✓
World Wide Web Consortium (W3C)		
Solution Install Schema	✓	
New standards to be developed	✓	✓

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.

Recent Standards Developments
One significant development in the area of autonomic computing management standards was the March 9, 2005 announcement by OASIS of the ratification of the WS-DM 1.0 specification (noted in Table 2). According to that announcement:

> OASIS, the international e-business standards consortium, today announced that its members have approved Web Services Distributed Management (WSDM) as an OASIS Standard, a status that signifies the highest level of ratification. WSDM enables management applications to be built using Web services, allowing resources to be controlled by many managers through a single interface.

The WS-DM 1.0 specification, available from OASIS, consists of two major parts: management using Web services (MUWS) and management of Web services (MOWS). The specification addresses a broad array of management topics relevant for autonomic computing, including properties, operations, events, capabilities and management interfaces. These WS-DM management topics can be realized in touchpoint manageability interfaces, using sensor and effector interfaces, described in Chapter 3.

Although Web services are not the only method for accomplishing autonomic computing, as indicated earlier, they provide a standard basis for management interfaces, and so the WS-DM 1.0 specification offers an important standards basis for constructing autonomic systems.

Another significant development related to autonomic computing standards is the launch of the OASIS Solution Deployment Descriptor technical committee. According to this new committee's call for participation at http://lists.oasis-open.org/archives/sdd/200504/msg00000.html this committee will "will define XML schema for [Solution Deployment Descriptors (SDDs)], as well as a package format to associate SDDs, resource content, and software artifacts. SDDs are intended to describe the aggregation of installable units at all levels of the software stack. The resulting XML schema shall be partitioned to allow for layered implementations covering the range of applications from the definition of atomic units of software (Smallest Installable Units) to complex, multi-platform, heterogeneous solutions."

This initiative is intended to result in an industry standard in the area of solution topology knowledge as described in Table 2 in Chapter 3.

6. Summary

Autonomic computing is about shifting the burden of managing systems from people to technologies. When the Self-Managing Autonomic Technology and 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 systemwide 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 an enterprise service bus (one of the five architectural building blocks) that integrates the remaining four architectural building blocks: autonomic managers, manual managers, touchpoints and knowledge sources.

Autonomic managers and manual 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). The journey to a fully autonomic IT infrastructure is an evolution. The Autonomic Computing Adoption Model offers a mechanism for characterizing autonomic maturity in three dimensions: the degree of automation within a given scope (in five stages, from fully manual to fully autonomic), the scope within which automation is applied (also in five stages, from a sub-component to an entire business system) and the IT management processes that can be automated (such as incident management, change management, and others that offer a model in which various tasks and activities within the process can be automated).

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 United States

Visit us at **ibm.com**/autonomic