Autonomic Computing
White Paper

An architectural blueprint for autonomic computing.

June 2006
Fourth Edition

Contents

Highlights

In this updated architectural blueprint, we describe not only the latest advances in the architecture and standards but also how the architecture can be applied in solutions that can achieve real business value.

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, and the complexity of the relationships among these components, 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 and service delivery. Human intervention is required to manage 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. Modern IT infrastructures are composed of 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.

All of this impedes effective and efficient execution of business processes and the delivery of services that rely on the IT infrastructure. In particular, the business of IT management has become too complex and costly.

Highlights

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.

Previous versions of this blueprint have described:

- The overall autonomic computing architecture, including architectural building blocks.
- The autonomic computing adoption model.
- The process of delegation from humans to automation.
- New developments in standardization.

This is not an exhaustive list of how the autonomic computing architecture can be used, but it provides illustrations of how business value can be achieved by exploiting the autonomic computing architecture.

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. This blueprint focuses on the application of these self-managing capabilities within self-managing resources and as part of ITSM solutions.

Highlights

A self-managing system could automatically deploy a new resource (such as a new server), install the appropriate software and tune the server's configuration for its intended usage. This is a notable shift from traditional processes that require a significant amount of manual analysis before and after deployment to ensure that the resource operates effectively and efficiently.

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, helping improve responsiveness and quality of service, reduce TCO, and enhance time to value through:

- *Selective process automation*–Typically, implementing these processes requires multiple IT professionals to initiate a process, collect incident details, open problem records, create requests for change, and perform numerous other activities. These activities often are based on incomplete or incoherent data and poorly documented processes. Self-managing capabilities help to integrate and relate the data that IT professionals use for decision making to improve the understanding and regularity of system management processes. In a self-managing system, components can perform certain process activities 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 execution of the process, and more accurate data from the system.

- *Reduced time and skill requirements*–IT management 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 many of the tasks that make up these activities can be encoded within the system so that 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 can be 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 infrastructure more efficiently with less human intervention, which can decrease costs and enhance the organization's ability to react to change. For instance, a self-managing system could automatically deploy a new resource (such as a new server), install the appropriate software, and tune the server's configuration for its intended usage. This is a notable shift from traditional

Highlights

Today's large-scale computing systems are amazingly complex, and require daunting expertise and patience to get them running and keep them running. The goal of autonomic computing is to dramatically reduce the cost and perceived complexity of these systems by enabling them to manage themselves in accordance with high-level guidance from humans.

processes that require a significant amount of manual analysis before and after deployment to ensure that the resource operates effectively and efficiently. This shift can enhance IT service delivery and eases the burden of IT management.

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 system-wide. For this to happen in a multi vendor infrastructure, vendors must agree on standards for autonomic systems. Later, we describe these standards in more detail and offer example applications of autonomic computing for self-managing resources and ITSM that demonstrate the critical role of standards in achieving self-managing 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 makes the transition from the theoretical to the practical view by presenting two example applications of the autonomic computing architecture: self-managing resources and ITSM. It also presents the autonomic computing adoption model that shows how self-managing autonomic capabilities can be achieved in an evolutionary manner and describes industry standards initiatives that are necessary to realize these applications of autonomic computing within an open system architecture for heterogeneous environments.

2. Autonomic computing architecture

Today's large-scale computing systems are amazingly complex, and require daunting expertise and patience just to get them running and keep them running. There is a legitimate concern within the IT industry that the increasing difficulty of system administration will become a barrier to deploying and maintaining large computing systems. The goal of autonomic computing is to dramatically reduce the cost and perceived complexity of these systems by enabling them to manage themselves in accordance with high-level guidance from humans. This has been recognized as a grand challenge—one upon which the future of information technology rides. Ultimately, success in meeting this challenge will depend not only on

Highlights

Our approach to autonomic computing builds upon well-established principles of distributed computing and systems management; although it involves new ways of addressing many of the issues, it is an evolutionary approach.

invention of new technologies, but also upon an architecture for self-management that exploits these technologies appropriately. Furthermore, the right architecture can provide the key to achieving autonomic behavior at the system level.

It is important to note that creating self-managing systems of the kind we describe here does not require achieving human-level intelligence in software, of giving computers human-like "common sense," or otherwise solving any of the deep problems of artificial intelligence. Our approach to autonomic computing builds upon well-established principles of distributed computing and systems management; although it involves new ways of addressing many of the issues, it is an evolutionary approach, as described in Chapter 4.

An architecture for autonomic computing must accomplish three fundamental goals:

- First, it must describe the external *interfaces* and *behaviors* required of individual system components.
- Second, it must describe how to compose these components so that the components can cooperate toward the goals of system-wide self-management.
- Finally, it must describe how to *compose* systems from these components in such a way that the *system as a whole is self-managing*.

Autonomic computing system

This blueprint organizes an autonomic computing system into building blocks that can be composed together to form self-managing systems. These building blocks can be 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:

- Manageability endpoints (standard manageability interfaces for managed resources, sometimes previously called *touchpoints*)
- Knowledge sources
- Autonomic managers
- Manual managers

These building blocks are the architectural representations of the components in an autonomic system and they work together to provide self-managing capabilities. One such composition is illustrated in Figure 1 and described next; each of the five building blocks is then detailed. Note

that Figure 1 shows only one topology for composing the building blocks; in this case, we illustrate a generic hierarchical arrangement of autonomic and manual managers. Other compositions are possible, depending on the services to be delivered and the organization's needs.

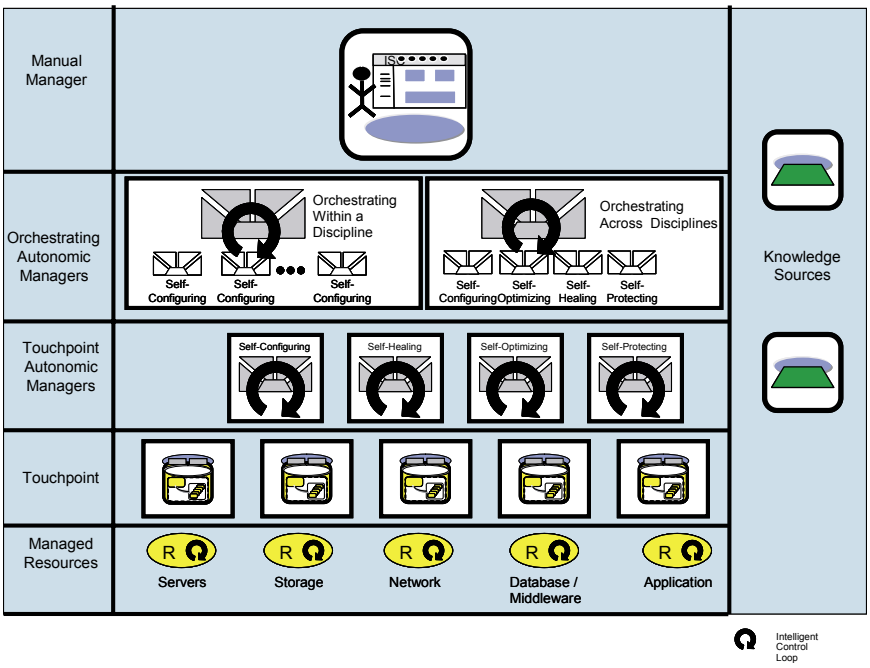


Figure 1. Autonomic computing reference architecture

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 (this latter aspect is detailed in Chapter 3). The next layer incorporates consistent, standard manageability interfaces for accessing and controlling the managed resources. These standard interfaces are delivered through a *manageability endpoint*. Layers three and four automate some portion of the IT process using an autonomic manager.

A particular resource may have one or more r*esource autonomic managers*, each implementing a relevant control loop. Layer 3 in Figure 1 illustrates this by depicting an autonomic manager for four broad categories of self-management (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 offer one way to deliver system-wide autonomic capability by incorporating

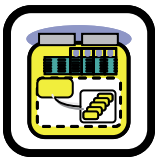
Highlights

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.

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.

Resources and managers may be composed to deliver particular services and execute business and IT management processes. Chapter 3 discusses the application of the autonomic computing architecture building blocks to ITSM, which is one example of such an arrangement. The remainder of this chapter describes these architectural concepts and the function of each of these parts of an autonomic computing system.

Manageability endpoints

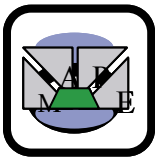


A manageability endpoint (sometimes previously called a touchpoint) is the component in a system that exposes the state and management operations for a resource in the system. An autonomic manager communicates with a manageability endpoint through the manageability interface. A manageability endpoint is the implementation of the *manageability interface* for a specific manageable resource or a set of related manageable resources. For example, a manageability endpoint might be implemented that exposes the manageability for a database server, the databases that database server hosts, and the tables within those databases.

The manageability interface for monitoring and controlling a managed resource is organized into its *sensor* (used to obtain data from the resource) and *effector* (used to perform operations on the resource). A key aspect of the autonomic computing architecture is the use of standard manageability interfaces. *Web Services Distributed Management* (WSDM, discussed in Chapter 5) is one such standard that can be used by manageability endpoints and autonomic managers.

Highlights

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 an intelligent control loop. 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 2, 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.

Highlights

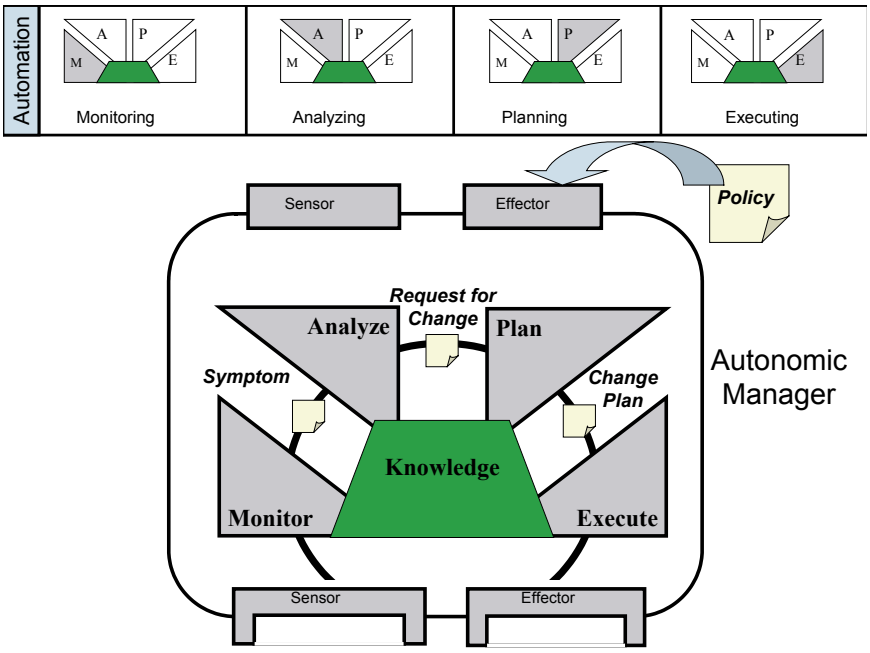


Figure 2. Functional details of an autonomic manager

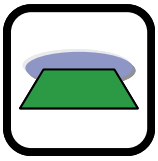
As illustrated in Figure 2, autonomic managers, in the same manner as manageability endpoints, provide sensor and effector interfaces for other autonomic managers and other components in the distributed infrastructure to use. Using sensor and effector interfaces for the distributed infrastructure components 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 resource autonomic managers to accomplish its management functions, as illustrated previously in Figure 1. This composition is discussed further in Chapter 3, "Self-Managing Resources" section.

Even though an autonomic manager is capable of automating the monitor, analyze, plan, and execute parts of the loop, IT professionals might delegate only portions of the potential automated functions to the autonomic manager. In Figure 2, four profiles (monitoring, analyzing, planning, and executing) are shown. An administrator might delegate only the monitoring function to the autonomic manager, choosing to have the autonomic manager provide data and recommended actions that the administrator can process. 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 delegation choices could allow additional parts of the control loop to be automated.

Highlights

In an autonomic system, knowledge consists of particular types of management data with architected syntax and semantics, such as symptoms, policies, requests for change, and change plans.

Knowledge source

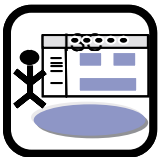


A knowledge source is an implementation of a registry, dictionary, database or other repository that provides access to knowledge according to the interfaces prescribed by the architecture. In an autonomic system, knowledge consists of particular types of management data with architected syntax and semantics, such as symptoms, policies, requests for change, and change plans. This knowledge can be stored in a knowledge source so that it can be shared among autonomic managers. In an ITSM environment, the *configuration management database* (CMDB) functions as a knowledge source that can contain this management data as well as many other types of knowledge, such as resource identification, details and relationships.

The knowledge stored in knowledge sources can be used to extend the 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).

Data used by the autonomic manager's four functions (monitor, analyze, plan, and execute) are stored as knowledge that could be shared among autonomic managers. The knowledge includes data such as topology information, historical logs, metrics, symptoms, and policies. In an ITSM environment, many types of shared knowledge can be stored in the CMDB.

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 or orchestrate autonomic managers and other manual managers (that represent other IT professionals). The manual manager building block is the architectural representation of the human activity and typically involves a human using a management console.

A manual manager can enable an IT professional to delegate management functions to autonomic managers, as described earlier.

A manual manager provides a common system management interface for the IT professional using an integrated solutions console. Although specific contents and user interfaces will vary by user roles and management processes, self-managing autonomic systems can use common console technology to create a consistent human-facing interface for the 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 by using a common console framework that exploits industry standards and promotes consistent presentation to IT professionals. Administrative console functions range from setup and configuration to solution run-time monitoring and control. In IT service management environments, the common console aids in managing the tasks and activities associated with service management and service delivery by aggregating information associated with the task or activity, such as knowledge from the CMDB and other knowledge sources, operational state, policies and operational procedures, in a single integrated view. This aggregation then helps the IT professionals select which tasks and activities can be delegated to autonomic managers.

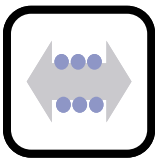
The customer value of an integrated solutions console can result in reduced cost of ownership—attributable to more efficient administration—and shorter learning curves as new products and solutions are added to the autonomic system environment. The shorter learning curve can be achieved by using standards 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, which helps reduce 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.

Highlights

In IT service management environments, the common console aids in managing the tasks and activities associated with service management and service delivery by aggregating information associated with the task or activity, such as knowledge from the CMDB and other knowledge sources, operational state, policies and operational procedures, in a single integrated view.

Enterprise Service Bus



An enterprise service bus is an implementation that assists in integrating other building blocks (for example, autonomic managers and manageability endpoints) 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 includes interactions such as:

- aggregating multiple manageability mechanisms for a single manageable resource.
- facilitating single or multiple autonomic managers to manage single or multiple manageability endpoints.

3. Using the autonomic computing architecture

The autonomic computing architecture has been described in previous versions of this blueprint and has been updated in this version. The architecture provides the foundation for self-managing autonomic solutions. To demonstrate the value of the autonomic computing architecture, we next describe two such solutions—self-managing resources and IT service management. Both of these examples show how the architecture can be realized in solutions that offer business value.

The applications of the architecture described next are but two examples of its use; the autonomic computing architecture can be employed throughout the IT infrastructure and business processes of an enterprise.

Self-managing resources

Three fundamental architectural goals were outlined in Chapter 2:

- Interface and behavior definition
- Composition
- System-wide self-management

To achieve these goals, we have identified several important architectural requirements.

The architecture should not impose requirements on the internal structure of individual components, but rather should focus on defining interfaces and behavior for these components. Autonomic managers and human system administrators should interact with individual components by reading and setting high-level policies, rather than by reading and setting the thousands of configuration parameters on today's components. In response to this high-level direction, the components should be substantially more self-managing than they are today. When they are first brought into a system, they should configure themselves internally. They should optimize their internal behavior in response to changing external conditions. They should heal themselves internally if problems occur. And they should protect themselves from external probing, attack, and corruption. This desired behavior should be managed through standard manageability interfaces prescribed by the architecture. Web Services Distributed Management (WSDM, described in Chapter 4) is an exemplary such interface. By focusing on defining the behavior and interfaces for components, the architecture does not prescribe any particular implementation or internal component structure, yet enables interoperability in heterogeneous environments.

Self-managing components also aid in simplifying composition. Today, getting a database to use a storage system involves detailed, manual configuration of both the database and the storage system. A system administrator must tell the database which particular storage system to use, where it is, and how to authenticate itself to the storage system, whereas the storage system must be told that the database is allowed to access it. In an autonomic computing system, on the other hand, components should compose themselves locally—the database should be able to find a storage system that it can use, bind itself to that storage system, and start using it—without human intervention, and without the need for a global configuration service that knows how to compose all possible combinations of components. Human administrators might set the policies for the database and storage system, but need not be involved in the mundane details of operationally establishing the component composition according to the policies.

Our final task is to simplify the management of entire systems. Ideally, it should be possible to compose our self-managing components into self-managing systems—systems that configure themselves, optimize themselves, heal themselves and protect themselves. It should be possible to treat an entire system—say, a payroll system, or a weather prediction system—as a single autonomic component. Ultimately, such a system should be able to accept a high-level policy and manage itself with minimal human operational intervention.

Highlights

It should be possible to treat an entire system—say, a payroll system, or a weather prediction system—as a single autonomic component. Ultimately, such a system should be able to accept a high-level policy and manage itself with minimal human operational intervention.

Next, we detail the considerations for self-managing resources as an application of the autonomic computing architecture.

Self-managing resources

In the autonomic computing architecture, self-managing resources manage their own behavior in response to higher-level goals, and interact with other resources to provide or consume computational services. They know themselves—what they are (an application server or a database, for example) and what version and release they are. They know about service agreements that they have with other resources (so, for example, a database knows which storage system it uses). They are responsible for configuring themselves, for healing internal failures, for optimizing their own behavior, and for protecting themselves from external probing and attack. To simplify systems management in the large, self-managing resources handle problems locally whenever possible, while still providing standard manageability interfaces to participate in system-wide management for those aspects for which the self-managing resource cannot, or is not best suited to perform.

Self-managing resources can include computing resources such as databases, storage systems and servers; system management components such as workload managers and provisioning subsystems; and administration through which an IT professional communicates with other self-managing resources. Self-managing resources also might include resources that assist others in doing their tasks, such as a policy repository, a sentinel, a broker and a registry.

In addition to standard interfaces for manageability and self-management, interoperability requires standardization of the vocabularies that self-managing resources use when communicating.

Policy

Of central importance to autonomic system behavior is the ability for high-level, broadly-scoped directives to be translated into specific actions to be taken by resources. This is achieved through the use of policies.

Policy interfaces enable administrators to establish new policies for a self-managing resource and to determine the policies currently in use by the resource. The ability to send new policy to a resource might involve only limited administrative functions (allowing, for example, only new monitoring or alert policies to be accepted), or it might involve wholesale, resource wide changes, replacing or adding any policy that the resource understands.

Highlights

For a system to be self-managing, its constituent components (resources, managers and knowledge sources) must be able to discover each other, identify other components with which to communicate, and coordinate with those other components to achieve their mutual goals.

Composing self-managing resources

Having described self-managing resources, we next seek to simplify the process of composing them. Composition of self-managing resources is accomplished by establishing policies that govern the resource's behavior. These policies might specify goals, objectives, profiles or agreements, which are explicit representations of the expectations that one resource has on another. For example, a database resource might need to use a storage resource; hence, it requires that the storage resource provide access to a certain amount of persistent storage for a known period of time. Similarly, a workload manager might need to ensure that a server keeps some of its operational parameters within a particular range until explicitly told to change them.

Policies make explicit the expectations that one resource has on another, thereby allowing the resources to reason about the goals, objectives, and agreements and to determine what actions they must take to satisfy these agreements. This is vital if dynamic, self-managing resources are to coordinate their behavior and work together to form larger, compound entities.

Agreements express the structure of an autonomic system—defining which resources use which other resources, and for what purposes. Many agreements will be left in place for relatively long periods of time—days, months, or even years—although, of course, they can be changed as the needs of the system change. Within the structure defined by these agreements, operational interactions among resources carry out the day-to-day operation of the system. A database might make an agreement to use a storage system and then operate within that agreement, perhaps for quite some time. It is not necessary to establish a new agreement every time the database accesses the storage system.

A self-managing resource should be able to attain the service characteristics (performance, and so on) specified in the policies that have been established for it. This might involve setting requirements for any services that it needs to request from other resources or that it provides to other resources. This enables self-managing resources to participate in self-managing systems.

Self-managing autonomic systems

The arbitrary composition of self-managing resources does not guarantee self-management at the system level. For a system to be self-managing, its constituent components (resources, managers and knowledge sources) must be able to discover each other, identify other components with which to communicate, and coordinate with those other components to achieve their

Highlights

Autonomic computing is critical to ITSM, because the ultimate goal for ITSM is not just to define and execute best practice IT processes, but also reduce the complexity of IT management processes and enable tasks within those processes to be automated.

mutual goals. In addition, there are system-level behaviors that by their very nature cannot be performed by any single resource, such as meeting end-to-end service level objectives.

In general, assembling a self-managing autonomic system requires:

(1) A collection of self-managing resources that implements the desired function.
(2) Autonomic and manual managers that implement system functions that enable the required system-level behaviors.
(3) Knowledge sources that contain management data such as symptoms, policies and solution topology information.
(4) Design patterns that offer models for the structure and arrangement of components for system self-management.

Suppose, for example, that we wish to create a self-managing autonomic financial transaction system. First, we must collect an appropriate set of self-managing resources, such as a router, firewall, Web server, database and storage resources. Second, we need management functions that support the operation of the system—such as sentinels that monitor system performance (autonomic mangers with a system-wide scope) and brokers that facilitate interactions (enterprise service bus patterns). Third, we need corresponding knowledge that supports those management functions, such as registries that allow resources to find one another and policies that specify performance goals. Finally, we require an appropriate set of design patterns to achieve system-wide self-management. These design patterns enable the composition of self-managing resources into a self-managing autonomic system.

IT Service Management

Our second example applies the architecture to the discipline of IT service management (ITSM). As described in [2]:

> *[The environment in which IT organizations face business pressures of change, compliance, complexity and cost] introduces challenges in managing IT cost and responsiveness across IT "silos" – vertical towers of specialized expertise and tools associated with managing one "slice" of the IT environment (such as servers, network, applications, databases, and so on). Managing composite applications and services in such silos is a struggle.*
>
> *IT service management (ITSM) is about integrating those silos – not only the technology, but also the people, processes and information*

> *associated with horizontal IT services such as availability management, change management, security management, incident management, and others.*

The same source also describes the role of autonomic computing in ITSM:

> *ITSM is the IBM initiative for defining and modeling the processes associated with IT management, including the incorporation of best practices based on the IT Infrastructure Library® (ITIL®). In ITSM, autonomic computing architecture and technologies are employed to provide management functions for the IT infrastructure, using standards-based management interfaces and data formats. Autonomic computing provides important IT operational management components of the IT management architecture… including management tools, resource management, user interface components, tooling and knowledge for the CMDB. Autonomic computing is critical to ITSM, because the ultimate goal for ITSM is not just to define and execute best practice IT processes, but to also reduce the complexity of IT management processes and enable tasks within those processes to be automated.*
>
> The remainder of this section details how the autonomic computing architecture is applied in ITSM solutions. First, we look at a simplified view of several typical IT processes and examine how tasks can be delegated from human administration to automation.

Delegated IT processes can deliver self-managing capabilities

In addition to the management functions that autonomic managers can perform when managing resources, the self-managing capabilities can be applied in other areas within the IT environment. 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 by ITIL and the IBM IT Process Model (developed by IBM Global Services). Figure 3 shows some simplified example process flows for incident management, problem management and change management (these simplified examples do not precisely match any particular process model, but rather show generic, illustrative representations of typical tasks in these processes). The more these tasks can be automated, the more opportunities will exist for IT professionals to delegate the management of the IT infrastructure to itself.

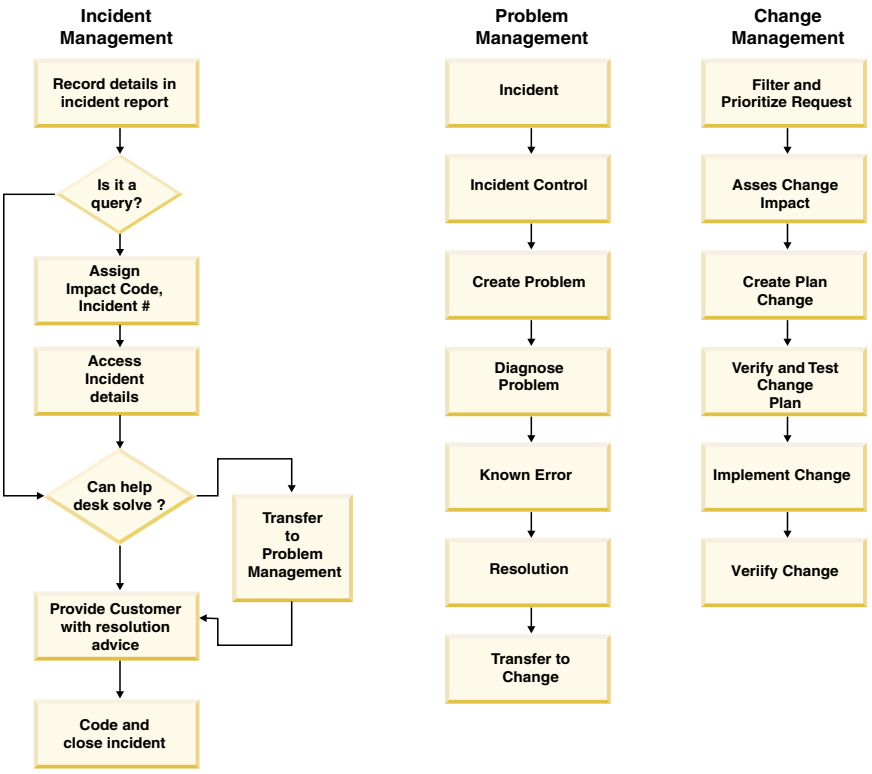


Figure 3. Typical IT processes

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 earlier) 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 3. One type of change that might be affected on a system is to provision new resources. The IT professional might choose to delegate 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. 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.

Autonomic computing architecture applied to ITSM solutions

Figure 4 illustrates an architecture for ITSM. As previously noted, autonomic computing provides important IT operational management components of ITSM, including management tools, resource management, user interface components, knowledge for the CMDB, and tooling. These map directly

to the architectural building blocks introduced previously (autonomic manager, manageability endpoint, manual manager, and knowledge source respectively, along with tooling that is introduced in the figure). The icons used in Chapter 2 appear in Figure 4 to identify the architectural building

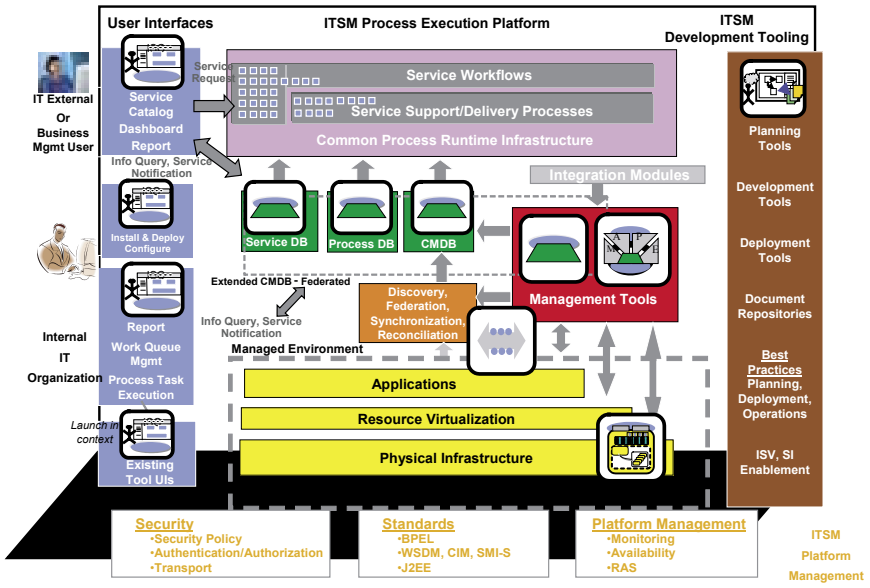
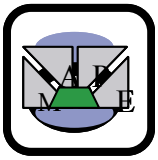


Figure 4. ITSM Architecture

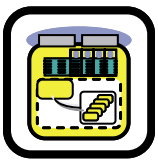
blocks associated with each of the corresponding ITSM subsystems. Next, we examine each of these subsystems in terms of the autonomic computing architecture, including the applicable architectural building blocks.

Management tools



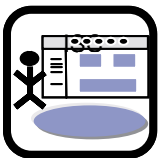
Operational management products interact with managed resources using standard manageability interfaces such as WSDM. The management tools incorporate full or partial autonomic control loops to enable the automation of tasks in IT management processes, using information in the CMDB, such as resource relationships and knowledge. Architecturally, these management tools take the form of autonomic managers.

Resource management



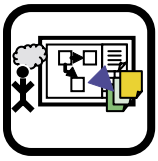
Resource configuration and relationship information is discovered, registered and populated in the CMDB. Resources use standard management interfaces such as WSDM to interact with management tools and the CMDB. The architectural building block associated with resource management is a manageability endpoint. Note that these resources could be self-managing resources as described earlier in this chapter.

User interface components



IT professionals can interact with all layers of the ITSM solution–resources, management tools and the CMDB–using a common console. This integration, via standards, enables consolidated information associated with IT management to be presented to the IT professionals in context, in an integrated manner, improving the efficiency of managing the IT processes and services. These human administration functions are filled by the role of the manual manager architectural building block.

Tooling

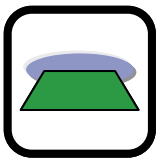


Development-time tooling can be used to incorporate manageability and management capabilities into the components that comprise the ITSM solution, building in the standards used for operational management of IT services and processes. The autonomic computing architecture does not define a separate tooling building block, but tooling is essential for developing the software that instantiates the products that take on the roles associated with all of the architectural building blocks.

Highlights

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.

Knowledge for the CMDB



Knowledge can be captured through human-driven or automated processes and stored as management data in the CMDB to improve the efficacy of IT service and process management. The CMDB is represented architecturally as a knowledge source.

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 IT professional 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.

Figure 5 depicts 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

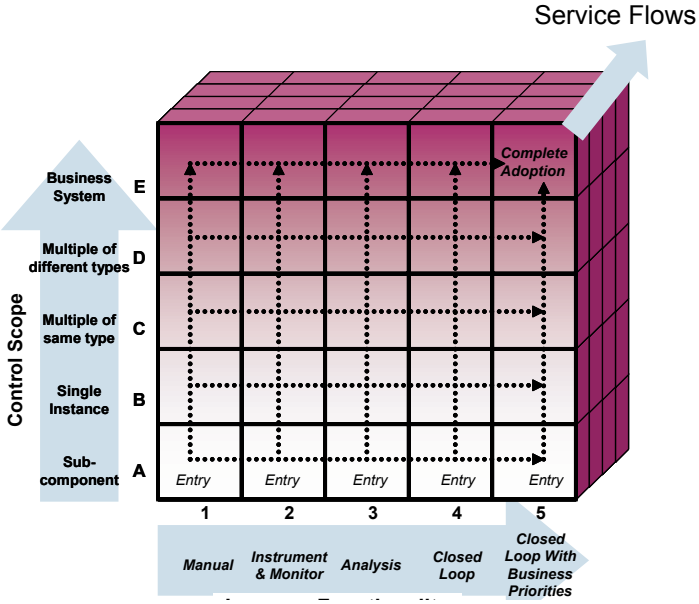


Figure 6. The autonomic computing adoption model

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 5, 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 manageability endpoints, 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 manageability endpoints. 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 5, correspond to the partial autonomic management functions shown in Figure 2.

The "control scope" dimension, along the y-axis of Figure 5, 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.

Highlights

Self-managing systems require various resources, managers, knowledge sources and other components from a diverse range of suppliers to be composed together. Therefore, these systems must be based on open industry standards.

- At the *multiple instances of different types* level, heterogeneous resources 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 5, captures the combination of IT management process activities that are being performed. These service flows 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 systems require various resources, managers, knowledge sources and other components from a diverse range of suppliers to be composed together. Therefore, these systems must be based on open industry standards. This blueprint identifies relevant existing computing industry standards that apply to self-managing systems. New open standards are being developed and will be developed in the future to define the mechanisms for interoperating in a heterogeneous system environment.

Examples of existing and emerging standards relevant to autonomic computing are provided in Table 1, followed by a discussion of recent developments in the area of new standards. A fuller discussion of many of these standards is included in [2].

Standards organization and standards
Distributed Management Task Force (DMTF)
Common Information Model (CIM), Web Services Common Information Model (WS-CIM)
CIM-Simplified Policy Language (CIM-SPL)
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)
Solution Deployment Descriptor (SDD)
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)
The Open Group
Application Response Measurement (ARM)

Table 1. Examples of standards related to autonomic computing

In addition to standards such as those listed in Table 1 (which comprise an illustrative, but not exhaustive list), other new standards are being developed and will be developed to support autonomic computing and self-managing systems. 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—for example, 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, in particular WSDM, for management and manageability interfaces. These techniques encourage implementers to leverage existing approaches and support multiple binding and marshalling techniques.

Recent standards developments

As illustrated in Table 1, many standards are applicable in self-managing autonomic systems. Several important new and emerging standards are discussed in more detail next.

WSDM

One significant development in the area of autonomic computing management standards was the March 9, 2005 announcement by OASIS of the ratification of the Web Services Distributed Management (WSDM) 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 WSDM 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 WSDM management topics can

be realized in endpoint manageability interfaces, using sensor and effector interfaces, described in Chapter 2.

The WSDM version 1.1 specification is expected to be ratified in July 2006. This update to the initial WSDM specification incorporates numerous enhancements, including new properties to enhance the semantic richness of the WSDM Event Format.

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 WSDM specification offers an important standards basis for constructing autonomic systems.

SDD

Another significant development related to autonomic computing standards is the launch of the OASIS Solution Deployment Descriptor technical committee. According to this committee's charter at http://www.oasis-open.org/committees/sdd/charter.php, "The purpose of this Technical Committee is to define XML schema to describe the characteristics of an installable unit (IU) of software that are relevant for core aspects of its deployment, configuration, and maintenance. This document will be referred to as the Solution Deployment Descriptor (SDD). SDDs, previously described as IUDDs, also are described in http://www.w3.org/Submission/2004/04/.

This initiative is intended to result in an industry standard in the area of solution topology specification, which can be rendered as knowledge as described in Chapter 2.

CIM-SPL

Common Information Model – Simplified Policy Language (CIM-SPL) is a CIM-compliant language for expressing IT management policies, with a standards description underway in the Distributed Management Task Force (DMTF), http://www.dmtf.org.

This new initiative is intended to result in an industry standard in the area of policy specification, which can be rendered as knowledge as described in Chapter 2.

5. Summary

Autonomic computing is about shifting the burden of managing systems from people to technologies. When 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 (in the form of self-managing resources) 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, manageability endpoints, and knowledge sources.

Autonomic managers and manual managers communicate with managed resources through the manageability interface, in the form of a manageability endpoint, using sensor and effector interfaces.

This paper described an architectural approach for creating self-managing resources and composing them to form self-managing systems. The desired self-management behaviors of self-managing resources—self-configuration, self-healing, self-optimization and self-protection—are recommended best practices in their design. A self-managing resource governs its behavior according to established policies and exposes manageability interfaces so that it can participate in system-wide autonomic control.

This paper also described the application of the autonomic computing architecture in IT service management environments. The architectural building blocks can be composed together to enable the automation of tasks in an IT management process through delegation.

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).

Open standards are key to achieving the self-managing autonomic vision. The design and composition of self-managing resources, and the automation of IT service management tasks, all rely upon the use of open standards, including WSDM, SDD and CIM-SPL, to enable consistent management in a multi-vendor, heterogeneous environment.

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 can free IT staffs from detailed mundane tasks, allowing them to focus on managing business processes. Autonomic computing can be accomplished through a combination of process changes, skills evolution, new technologies, architecture, and open industry standards.

References

1. Organization for the Advancement of Structured Information Standards (OASIS), Web Services Distributed Management, Version 1.1 (specifications and related documents), available from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm#technical

2. IBM Corporation, IT Service Management Standards: A Reference Model for Open Standards-Based ITSM Solutions, April 2006, ftp://ftp.software.ibm.com/software/tivoli/pdf/itsmstandardsreferencemodel.pdf

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

Glossary

A

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. See also **autonomic manager**.

autonomic

Pertaining to an on demand operating environment that responds automatically to problems, security threats, and system failures. See also **autonomic computing**.

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. See also **self-configure**, **self-heal**, **self-optimize** and **self-protect**.

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.

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. See also **analyze function**, **execute function**, **monitor function** and **plan function**.

C

change management

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

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.

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. See also **integrated solutions console**.

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.

E

effector

An interface that enables state changes for a managed resource. Contrast with **sensor**. See also **managed resource**.

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.

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. See also **autonomic manager** and **effector**.

I

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.

integrated solutions console

A technology that provides a common, consistent user interface, based on industry standards and component reuse, that 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.

K

knowledge

Standard data shared among the monitor, analyze, plan and execute functions of an autonomic manager, such as symptoms and policies. See also **autonomic manager**.

M

manageability endpoint

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

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. See also **autonomic manager** , **autonomic computing** , **sensor** and **effector**.

managed resource

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

monitor function

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

O

orchestrating autonomic manager

An autonomic manager that works with other autonomic managers to provide coordination functions. See also **autonomic manager**.

P

plan function

The autonomic manager function that structures the actions needed to achieve goals and objectives. See also **autonomic manager**.

policy

A set of considerations that are designed to guide the decisions that affect the behavior of a managed resource.

Q

quality of service

A measure of system performance and system availability.

R

resource autonomic manager

An autonomic manager that works with managed resources through their manageability endpoints. See also **autonomic manager**.

S

self-configure

To adapt to dynamically changing environments.

self-heal

To discover, diagnose, and act to prevent disruptions.

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.

sensor

An interface that exposes information about the state and state transitions of a managed resource. Contrast with **effector**. See also **managed resource**.

W

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.

IBM Corporation
17 Skyline Drive
Hawthorne, NY 10532
U.S.A.
Published in the United States of America
06-05

G507-2065-00