IT service management architecture and autonomic computing

P. Brittenham
R. R. Cutlip
C. Draper
B. A. Miller
S. Choudhary
M. Perazolo

The IT Infrastructure Library® defines a set of best practices to align information technology (IT) services to business needs and constitutes the framework for IT service management (ITSM). This framework helps organizations manage their IT services using standard design patterns and the requisite customization. In this paper, we discuss critical contributions that autonomic computing offers to the definition and implementation of an ITSM architecture and infrastructure. We first introduce key architectural patterns and specifications of autonomic computing as they relate to an ITSM logical architecture. We then show how autonomic computing delivers value through a set of ITSM-based case studies that address problem determination, impact assessment, and solution deployment.

INTRODUCTION

Autonomic computing traces its beginnings to eight key elements, or *theses*, described in Dr. Paul Horn's, "Autonomic Computing Manifesto," first delivered as a keynote address to a National Academy of Engineers meeting in 2001.[1] In his presentation, autonomic computing was described as a grand challenge—not just within IBM, but for the information technology (IT) industry as a whole.

In 2003, an *IBM Systems Journal* issue focused on a broad set of technologies that represented the state of the art then for autonomic computing. In their introductory paper, "The Dawning of the Autonomic Computing Era," Ganek and Corbi[2] examined both marketplace and industry drivers for autonomic computing.

Since its inception, the autonomic computing initiative has witnessed considerable success within the commercial realm,[3–6] it has been widely embraced within the research community,[7–11] and it has been leveraged in conjunction with other key architectural initiatives, including grid and Web services, pervasive and ubiquitous computing, and service-oriented architecture (SOA).[12–14] Considerable activity also has occurred within the standards communities related to autonomic computing.[15,16]

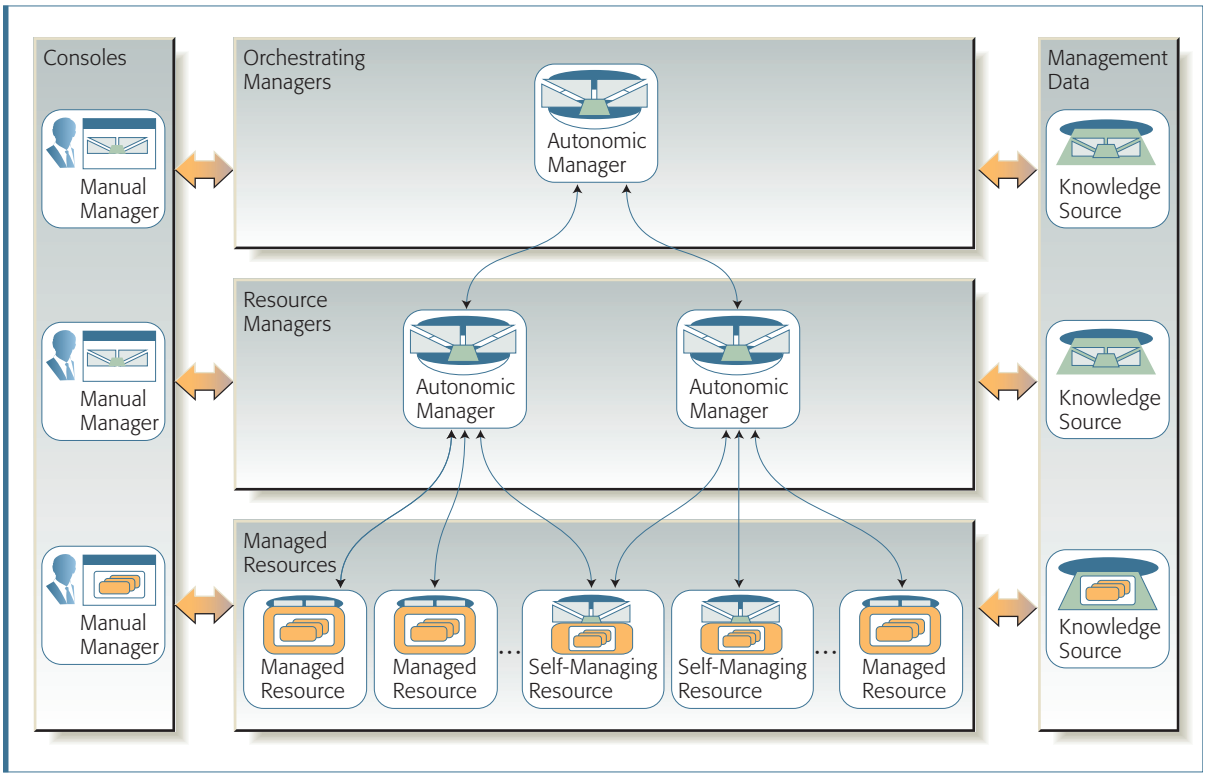

Figure 1
Autonomic computing reference architecture (ACRA)

In a broad sense, autonomic computing is a progressive evolution of architecture, technology, and standards that addresses IT complexity. This complexity is driven in large measure by increasing system-design and IT-management complexity, the increasing need for businesses to adapt quickly to compete, and the behavioral complexity spawned by an increasingly interconnected world.

This paper focuses on how autonomic computing addresses IT process efficiency that is largely predicated on the entities, processes, and disciplines described within the Information Technology Infrastructure Library** (ITIL**).[17] (For our purposes, IT process efficiency addresses efficiency in terms of cost and time as well as effectiveness, in that the processes address appropriate business objectives.[18]) More specifically, we examine key autonomic computing contributions to IT service management. Our intent is to highlight the current and future contributions of autonomic computing to IT service management from an architectural context, making use of case studies that automate ITIL-based processes. In addition, we provide IT architects with additional perspectives and insights for their service delivery needs, using these case studies. We offer additional context by briefly examining the relationship of the Autonomic Computing Reference Architecture, including derivative architectural patterns and the associated specifications, to the IT service management (ITSM) logical architecture.

AUTONOMIC COMPUTING REFERENCE ARCHITECTURE

The autonomic-computing-reference-architecture (ACRA)[19] conceptual view consists of three parts: a set of architectural elements for constructing autonomic systems, patterns for using these elements in a system context, and interface and data interchange specifications that facilitate integration.

As shown in *Figure 1*, ACRA provides a basic systems-management topology that includes a hierarchical set of managers which manage a set of resources. The orchestrating managers control the management operations of the resource managers, and the resource managers provide the management support for a set of resources. Both types of

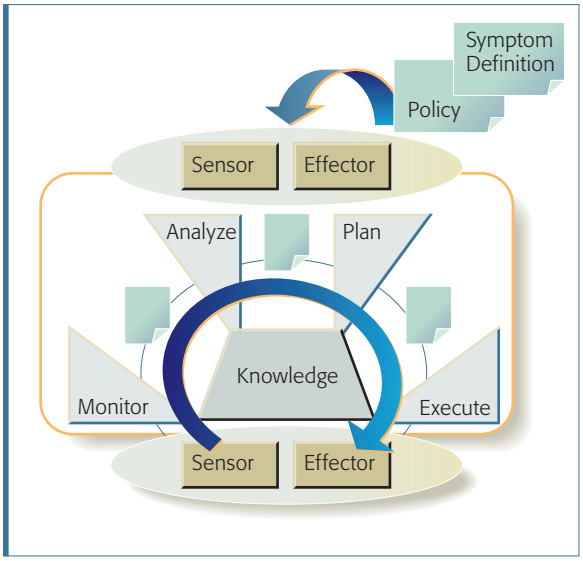

Figure 2
Autonomic manager

managers may implement autonomic manager capabilities and typically support user interaction through one or more manual manager elements. The managers might also access management data from one or more knowledge sources, as shown on the right side of Figure 1.

The managed resources shown in the bottom row of Figure 1 might or might not contain self-management capabilities (a resource that has self-management capabilities is a *self-managing resource*). Interactions between managers and resources may be direct or indirect (using agents) and are simplified by adopting a manageability standard, such as Web Services Distributed Management (WSDM)[20] or Common Information Model (CIM).[21]

The central component in the ACRA is the autonomic manager (***Figure 2***). It automates certain management functions and externalizes these functions according to the behavior defined by management standards.

An autonomic manager contains an intelligent control loop that implements four functions: monitor, analyze, plan, and execute. The monitor function collects details about the resources being managed. The analyze function takes the collected information and determines where changes are required. The plan function is responsible for generating any required plans, and the execute function takes necessary actions to implement planned changes.

The external interfaces for an autonomic manager provide a standard method to access the functions it supports. There are two logical external interfaces; both contain sensors and effectors. The interface that is logically at the bottom of the autonomic manager is used to interact with the resources that it manages, such as obtaining data from the resource through the sensor interface or performing operations on the resource through the effector interface. The interface that is logically on top of an autonomic manager is used by other managers to obtain information from the autonomic manager by means of a sensor and configure its autonomic capabilities by means of an effector, in much the same way that the autonomic manager interacts with its managed resources. For example, the effector interface can be used to set the policies the autonomic manager should adhere to or to set the symptom definitions that it should detect (a symptom definition is used to identify a possible problem or situation in the managed environment and is described in more detail later).

Autonomic managers are self-managing and manage their own behavior by using policies.[22] In addition, the autonomic manager makes use of knowledge sources to access management data, such as policies and symptom definitions, that are used to carry out its management functions.

In addition to autonomic managers, the ACRA also includes other elements, as shown in Figure 1:

- *Manual manager*—An implementation of the user interface that enables an IT professional to perform some management functions manually.
- *Managed resource*—System components that make up the IT infrastructure. These components make the state and management operations for a resource accessible.
- *Self-managing resources*—A type of managed resource that includes an integrated intelligent control loop.
- *Knowledge source*—Implementation of a registry, dictionary, database, or other repository that provides access to knowledge according to the interfaces prescribed by the architecture.

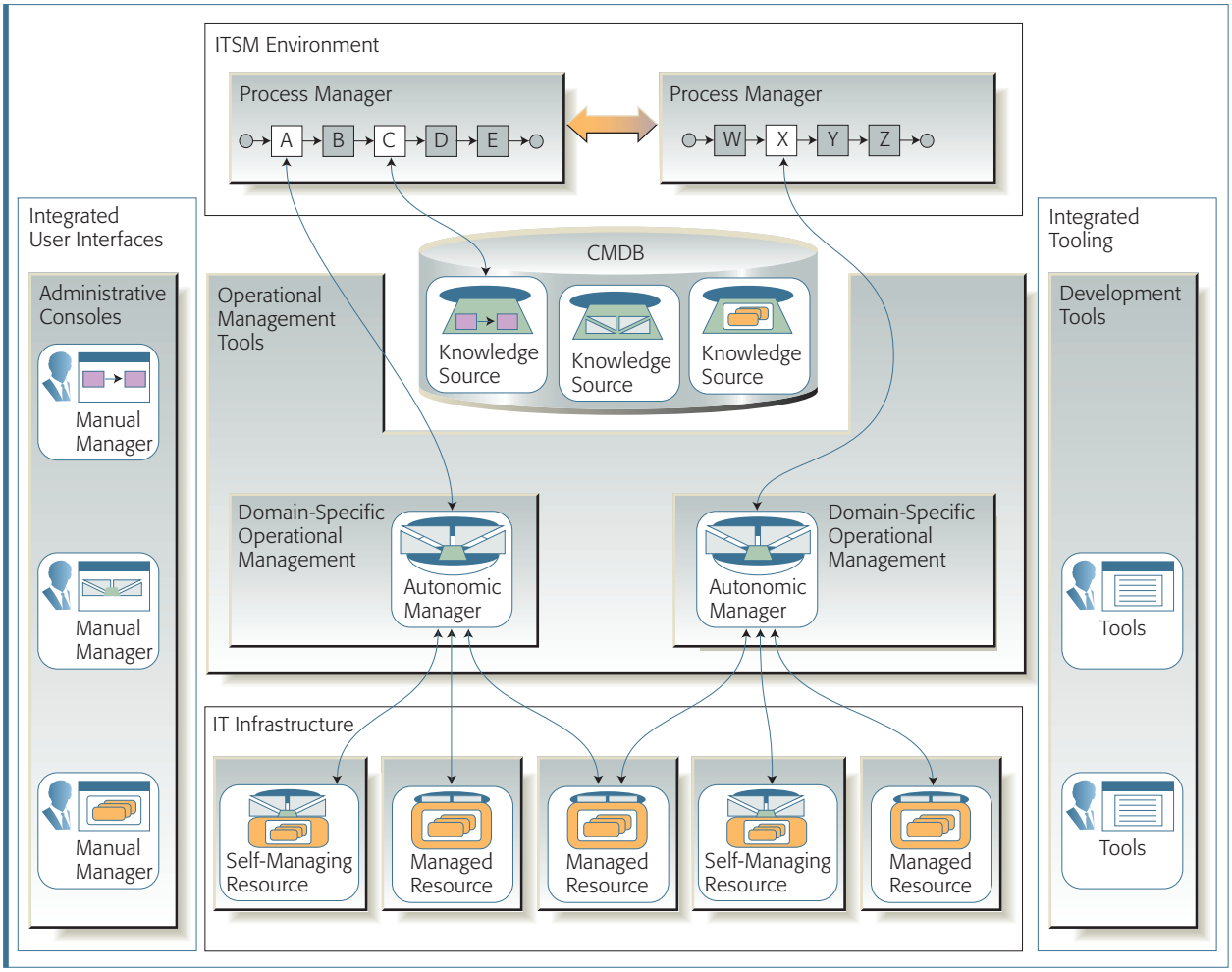

Figure 3
Relationship of ACRA and ITSM

The relationship between ACRA and ITSM is presented in ***Figure 3***. At the top is the layer composed of ITIL-based processes, such as those associated with change management, configuration management, problem management, and availability management. These processes consist of workflows that are composed of activities, which in turn are composed of tasks. Particular tasks may interact with domain-specific operational management tools that ultimately interact with both the managed resources and self-managing resources within the infrastructure layer.

Both managed resources and self-managing resources, shown at the bottom of Figure 3, exist within the IT infrastructure layer. In the context of these processes, the configuration management database (CMDB) assumes the role of a knowledge source as defined in the ACRA.

On the left side of Figure 3 is a set of user interfaces for various user roles that interact with the ITSM environment, operational management tools, and IT infrastructure layers. Here one would expect to find administrative tools that realize the ACRA manual-manager architectural element. On the right side is the integrated tooling layer, including development tooling, that interacts with all three levels of the logical architecture and with the CMDB.

Next, we examine the architectural patterns that derive from the architectural elements and support the requisite specifications and standards.

Autonomic computing architectural patterns

This section describes common patterns for using autonomic computing principles. Not all architectural patterns associated with autonomic computing are described here; we address only those patterns that are used in the ITSM case studies presented later.

Partial autonomic manager

The internal and external interfaces for an autonomic manager allow it to support partial implementations of the autonomic control loop. This is important because existing operational management tools can be used in the control loop by implementing the appropriate logical interfaces for the functions they provide (monitor, analyze, plan, and execute) as well as the standard external interfaces that allow access to those functions. This means that a single product or component need not implement the entire control loop. It can be composed together with other products or components to form a complete loop.

In addition, the logical interfaces between the functions in the control loop pass specific types of knowledge. For example, the monitor function collects the details from resources and organizes them into *symptoms* (described later) that need to be analyzed. If changes are required, the analyze function passes a *change request* to the plan function. The change request describes the modifications that the analyze component deems necessary or desirable in terms of the result. The plan function passes the appropriate change plan to the execute function. (The *change plan* represents a required set of changes for the manageable resource.)

Support for partial control loops enables autonomic managers to reflect customers' organizational structures, perhaps through alignment by management discipline, such as change management or problem management. Partial autonomic managers can be reused in multiple contexts. For example, a planning-and-execution engine for change management might be driven by multiple monitoring-analysis engines. Partial autonomic managers support the incremental delivery of self-managing autonomic capabilities; for example, adding new symptom definitions that enable new autonomic capabilities for problem determination.

Delegation

Delegation defines a set of patterns for progressing from manual management to autonomic management, with the goal of reducing the complexity associated with managing IT systems. Delegation refers to the process of describing what IT professionals do when they assign the tasks for which they are responsible in terms understood by operational management products that have self-managing autonomic capabilities. For example, delegation can be used by an IT professional to set a policy that allows a specific type of change to be pre-approved, thus allowing a task to be completed without human intervention. When IT professionals delegate tasks, they decide to exploit a self-managing autonomic capability present in a management tool. This implies that they trust the autonomic technology to perform the task correctly without intervention; however, they can take back control of the delegated task whenever they decide it is necessary. An IT professional must be responsible for the task before he or she can delegate it.

As shown in *Table 1*, there is a continuum associated with delegating tasks from manual processing (a task managed by an IT professional) to autonomic processing (a task executed automatically without direct interaction with an IT professional). Multiple levels of automation exist between these two extremes.

The ability to reverse the delegation of a task is an important factor in building trust in autonomic systems. This ability allows the control of a delegated task to be taken back at the discretion of an IT professional, because it is conceivable that a delegated task might not have the desired effect on key system metrics. In addition, the delegated task should be able to present options and recommendations to the administrator and to record the decisions it has made and the actions it has taken. This allows an administrator to verify the results of the delegated tasks so that trust in the delegation patterns can be built over time.

Self-managing resources

A *self-managing resource* integrates certain autonomic management capabilities into the resource. A self-managing resource contains a control loop responsible for managing those entities within its domain of control. This is distinct from a manage-

Table 1 Task delegation options

Delegation Option	Description
Manual processing	Processing with no automation.
Automated assistance	Automated tools help the IT professional perform the task, for example, by querying or processing data. The tool does not perform a complete autonomic function; the professional does not delegate responsibility.
Supervised delegation	An autonomic manager provides a recommendation to the IT professional, who must approve it before the task can proceed. Visual notifications that allow for partial delegation are provided by the manual manager.
Conditional delegation	The IT professional trusts an autonomic manager to perform some but not all requests. Whether or when to delegate a task may be based on meeting specific conditions that are defined in policies or rules.
Task delegation	The IT professional trusts an autonomic manager to perform a complete task.
Full-loop delegation	The delegated function consists of a full control loop that proceeds without manual intervention in normal operation.

ment tool that provides external autonomic management for a set of resources.

An application server, database server, and storage server could all be self-managing resources. For example, self-managing autonomic capabilities could be embedded in a database server to allow it to detect situations within its domain and to take actions to correct errant situations in that domain without requiring intervention from a management tool.

A benefit of self-managing resources is the ability for the resource to manage events and resource state data that affect its operational abilities, including performance and availability needs. Self-managing resources also reduce the overall system complexity by handling some of the details within the resource and by determining how to recover from certain problems without flooding the system-wide resource monitors and IT professionals with a plethora of information.

Self-managing resources may implement the partial autonomic manager pattern. This allows a self-managing resource to participate in delegation patterns. For example, a self-managing resource might provide a recommended action to an administrator, who could then approve that action or perform an alternate action. In addition, a self-managing resource could have management responsibility delegated to it by a higher-level management tool.

Hierarchical autonomic managers

Autonomic managers can be arranged in a hierarchy to reflect the way that IT professionals are organized. Although a hierarchy is not the only possible topology for management tools, this organization allows autonomic managers to focus on specific disciplines such as performance, availability, security, and others within a single domain of interest. Hierarchical autonomic managers provide a level of efficiency by allowing each autonomic manager to focus on its area of concern, but still provide the capability to build a complete autonomic system by composing the autonomic managers together in a hierarchy.

Hierarchical relationships between autonomic managers can also be used to orchestrate the services provided by a set of autonomic managers. For example, an autonomic manager that is responsible for the overall service-level objectives for a system might direct other autonomic managers to achieve objectives that are specific to their disciplines. If all of the discipline-specific autonomic managers meet their objectives, then the autonomic manager responsible for the overall system objectives will meet its objectives.

This hierarchical autonomic manager pattern is accomplished by using the external interfaces described earlier; an autonomic manager can manage or be managed by other autonomic man-

Table 2 Specifications for interfaces and data exchange formats

Specification	Description
WSDM[20]	Defines a standard manageability interface that can be used by autonomic managers and managed resources in a Web Services environment. WSDM is divided into two main parts: management of Web Services and management using Web Services.[23,24]
WSDM Event Format (WEF)	Defines a common format for representing information typically carried in events, including the source and time of the event, the situation that caused the event to be generated, and other associated data. The Common Base Event[25] is the IBM initial implementation of WEF.
Symptoms Reference Specification[26]	Describes how to define symptoms and how to represent them once they are recognized at runtime by a management tool. Symptom definitions are used by autonomic managers to recognize symptoms associated with monitored resources and determine what actions they should take or what recommendations they should propose.[27] An instance of a symptom is detected by correlating monitored data such as events, metrics, and resource state data.
Solution Deployment Descriptor (SDD)[28]	Defines the deployment characteristics of a software package. The SDD is used by the plan function and the execute function of an autonomic manager. It describes how to deploy software components at all levels of the IT stack by using a simple architectural pattern of deploying artifacts into target hosting environments to create, update, or configure resources.

agers in a manner similar to the way in which autonomic managers manage resources: using the sensor and effector of the external interfaces of the autonomic manager.

Autonomic computing specifications

Table 2 introduces the specifications for interfaces and data exchange formats that are relevant to the ITSM-based case studies that follow. It is intended only as a basic introduction; consult the cited references for a more detailed discussion of these and other specifications associated with autonomic computing.

ITIL PROBLEM-DETERMINATION CASE STUDY

This case study shows how capabilities embodied in the ACRA offer new advantages and additional value for several IT operational-services processes. The services are studied in the context of the IBM Tivoli* Unified Process (ITUP)[29] and consist of incident management, event management, and problem management. *Incident management* detects, records, classifies, investigates, diagnoses, and resolves incidents, including recovery actions. *Event management* identifies and prioritizes events and helps identify the responses to those events, which could cause incidents to be created. *Problem management* controls problems and known errors, manages problems proactively, monitors and reports problems, establishes a problem-management framework, evaluates problem-management performance, and determines root causes of incidents and problems.

This case study demonstrates the advantages of applying the delegation pattern and the partial autonomic manager pattern to the task of monitoring resources, automatically detecting and responding to disruptions, and assisting with problem diagnosis, all in an incremental and flexible way. The second scenario in this case study illustrates the benefits of using the hierarchical-autonomic-manager pattern for problem-determination knowledge sharing.

Problem-determination scenario 1—Operations automatically respond to a server outage

This scenario illustrates a server executing a critical customer-facing Web-based application that runs out of system resources, resulting in the application being unavailable. This problem has been occurring intermittently over the past few weeks, and the underlying cause has yet to be determined, although a workaround—reboot the server—has been determined. The operations staff has been asked to automate this detection and workaround to mini-

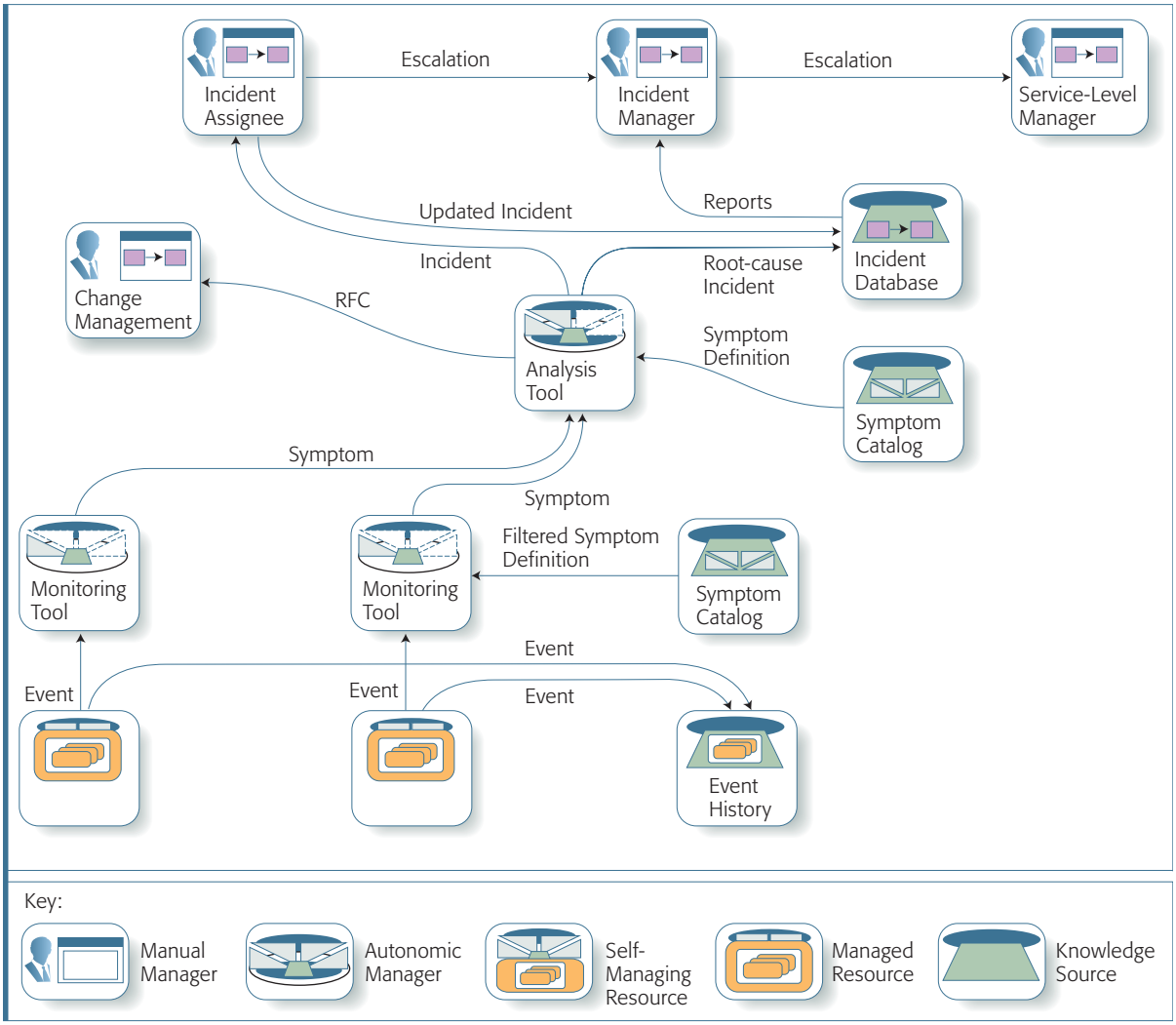

Figure 4
Scenario 1: Incident management

mize down time. A server providing a service begins to run out of system resources. Event-monitoring software detects, logs, examines, and filters server-disruption events. These events are correlated with events from other sources and escalated to an operational console.

Figure 4 illustrates how problem-determination scenario 1 utilizes the architectural elements of the ACRA. The set of partial autonomic managers (monitoring tools) that are monitoring the managed resources detects symptoms based on symptom definitions stored in a knowledge source. These autonomic managers have a hierarchical relationship with an autonomic manager (analysis tool) that is responsible for performing the detailed analysis of the symptoms that they detect. The analysis performed by this higher-level manager may result in a request for change (RFC) that will initiate the change management process, or it may result in an incident that will initiate the incident-management process. When an incident is created, the root-cause analysis data is stored in a knowledge source so that it can be referenced during the incident-management process.

The following sections contrast the activities in the current process with the autonomic process, which makes use of the capabilities defined in the ACRA. (The term *autonomic process* in this context refers to a process that has been enhanced to contain autonomic capabilities.) The capabilities of our

approach enable the incident-management activities to be more efficient and effective through increased automation.

Establish incident-management framework

In the current process, a human administrator determines the rules and procedures for incident management, based on the IT infrastructure and capabilities of management tools and human administrators. For the server, this includes the workaround (reboot the server) when the out-of-system-resources incident occurs. These policies are documented so that human administrators can apply them.

In the autonomic process, manual administration can be automated by determining the symptom definitions to be deployed in the management tool, which contain the automated workaround (reboot the server) when the out-of-system-resources incident occurs, along with the correlation patterns used to detect that incident.

Detect and record incident

In the current process, an incident record is opened, based on operator observations using event-monitoring tools.

In the autonomic process, an incident record is opened, based on monitoring event patterns as recorded in the production-level symptom definitions. Typically, simple single-domain symptom definitions (such as the example for rebooting the server) perform straightforward filtering of events. Common problem-pattern recording, as described in the ACRA and expressed in symptom definitions, enables rapid identification of intermittently occurring problems

Classify incident

In the current process, incident records raised in the previous activity are now analyzed by a human adminstrator to discover the reason for the incident. The incident should be classified by looking at known errors and problems and examining input parameters or assigning new parameters, such as impact, urgency, and priority. This process determines how further resolution actions are determined.

The autonomic process can automate incident classification for certain incidents (those for which appropriate symptom definitions exist). Once the symptom pattern is matched, additional information in the symptom definition provides incident classification information, including impact, urgency, priority, information about the resource associated with the incident, and additional information that is used to correlate this incident with others. Common incident classification recording, as described in the ACRA and expressed in symptom definitions, enables automated incident classification. This capability, in turn, takes advantage of a common event format, such as WEF, that facilitates automated event correlation and pattern matching to recognize the symptom.

This automation (and the automated functions described in subsequent sections) illustrates a pattern of delegation in which the human administrator is delegating the task—in this case, incident classification—to an autonomic manager.

Investigate and diagnose incident

Incident diagnosis in the current process must supply a human administrator with the means to continue business, perhaps with a degraded service. Temporary workarounds (reboot the server) are provided to minimize the impact of the incident on the business and to provide more time to investigate and devise a structural resolution.

In the autonomic process, single-domain symptom definitions can be correlated with other single-domain or cross-domain symptom definitions that associate problem resolution information with an incident record. The symptom associated with the incident can contain recommended remedial actions. In this scenario, a particular workaround (reboot the server) is identified in the corresponding symptom definition. Common-problem remedial-action recording, as described in the ACRA and expressed in symptom definitions, enables automated workarounds for intermittently occurring problems.

The use of monitoring for incident classification and analysis for incident diagnosis illustrates the use of the partial autonomic manager pattern. As shown in Figure 4, these functions of the intelligent control loop can be split between two distinct management tools.

Resolve incident and recover service

Workaround and recovery actions (reboot server) in the current process are carried out, often by specialized staff (second- or third-level support).

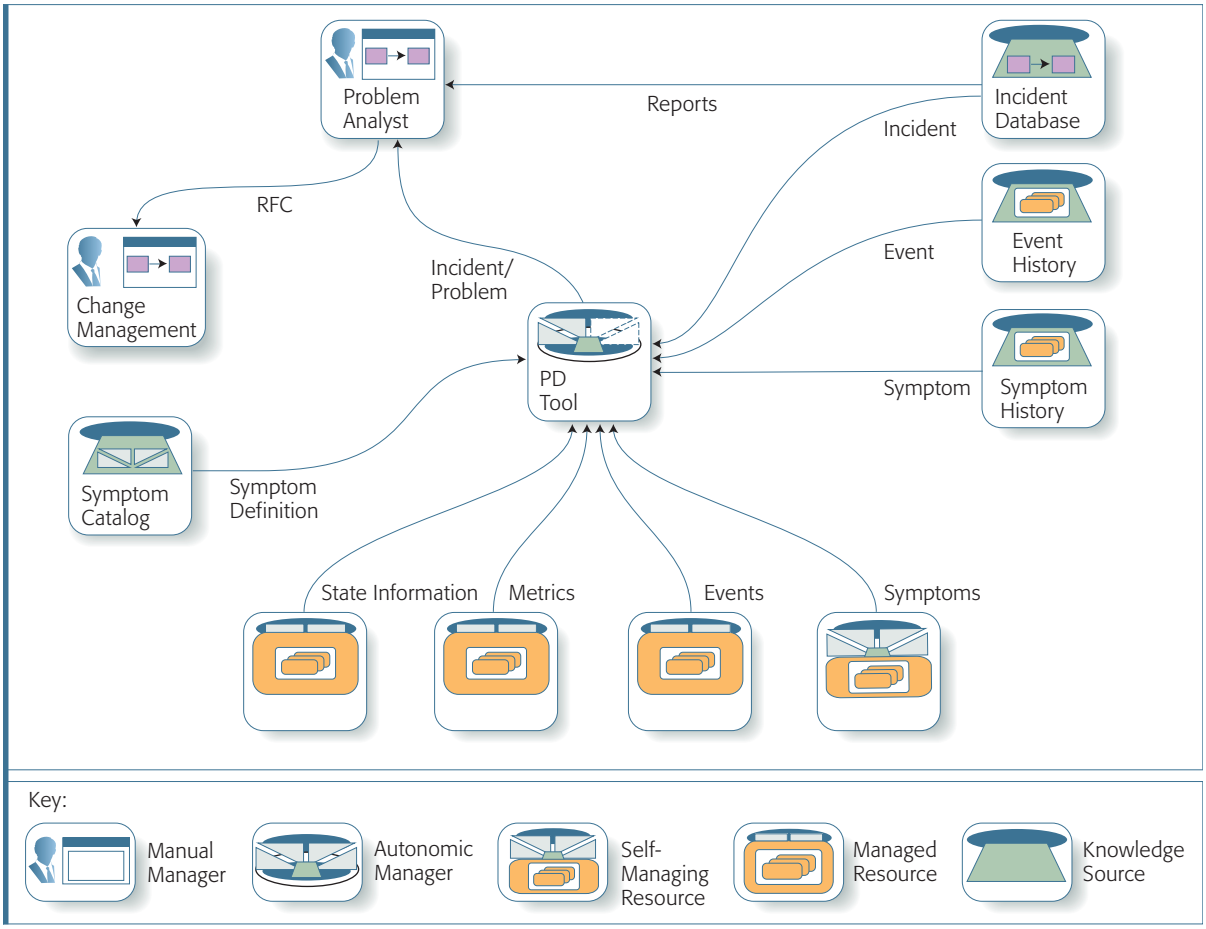

Figure 5
Scenario 2: Problem management

In the autonomic process, the automated response identified from the preceding diagnosis (reboot the server) is performed using programmatic control of the server. After the workaround is performed, event monitoring indicates that the server is correctly functioning again. Based on this, the incidents are closed. The ACRA describes a standard manageability interface, such as WSDM, that enables automated operations to be performed on the resource to accomplish the workaround.

Problem-determination scenario 2—Problem diagnosis

Scenario 2 of this case study is similar to scenario 1, but in this case, the identified automated response does not resolve the incident, so additional analysis is required to prevent the incident from happening again in the future. In this scenario, we expand scenario 1 by determining that the server software contains a memory leak, and that a software patch must be applied to correct the problem.

Figure 5 shows how the architectural elements of the ACRA are used in problem-determination scenario 2. In this scenario, a partial autonomic manager problem-determination (PD) tool performs PD by using state information, metrics, and events it receives from managed resources and self-managing resources. Because a self-managing resource could perform analysis based on symptom definitions, it also could generate symptoms to be analyzed by the PD tool. The analyses performed by the PD tool also use data from various knowledge sources. This includes symptom definitions and historical data, such as previously detected symptoms, events, and incidents. The symptom-based analysis performed by the PD tool may result in an RFC that is used to initiate the change management process.

The following sections contrast the current process and the autonomic process activities in the same manner as presented in scenario 1. The capabilities of our approach enable the problem-management activities to be more efficient and effective through increased automation.

Establish problem-management framework

In the current process, a human administrator determines the policies and procedures for problem management based on the IT infrastructure and capabilities of management tools and human administrators. For the server, this includes the list of known errors when the out-of-system-resources incident occurs, as well as additional analysis procedures to be followed for the investigation of new problems.

In the autonomic process, manual activities can be automated by determining the symptom definitions to be deployed in the management tool. These symptom definitions contain the RFCs to be applied when the out-of-system-resources problem occurs (in this case, apply a software patch) and the correlation patterns used to detect the problem.

Such correlation patterns are inherently more complex than similar incident detection patterns (described in problem-determination scenario 1) and often refer to configuration item (CI) configuration data and dependencies to effectively identify a problem. This activity is known as *root-cause analysis*.

Detect and record problem

In the current process, incidents are compared to known errors. If there are no known errors associated with the incident, a problem record is opened, based on operator observations using management tools.

In the autonomic process, a problem record is opened, based on correlating patterns from incident and event management, as recorded in symptom definitions for those processes. (Problem-determination scenario 1 describes the use of symptoms in incident management.)

Classify problem or assess error

In the current process, known errors are assessed, based on recorded information. Problem records raised in the previous activity are analyzed to discover the reason for the problem. The problem should be classified in a process similar to classifying incidents, described in scenario 1.

The autonomic process can automate problem classification. Once the symptom pattern is identified, additional information in the symptom definition provides problem-classification information, including impact, urgency, priority, information about the resource associated with the problem, and additional information that may be used to correlate this problem with others.

Investigate and diagnose problem

In the current process, the problem investigation activity is similar to that of incident investigation, but its primary objective is significantly different. The aim of incident investigation is to rapidly restore service, whereas the aim of problem investigation is to diagnose the underlying cause. Problem investigation often thoroughly analyzes CI configuration data and dependencies. In our example, problem investigation staff examine software resources deployed in the server, and upon diagnosis, determine that the underlying problem is a memory leak.

In the autonomic process, the problem-investigation activity is similar to that of incident investigation (described in problem-determination scenario 1), but more complex symptom definitions are used to determine the root cause of a problem. Typically, problem-related symptom definitions are processed by associating contextual information (such as CI configuration and dependencies), state data, events, and log records with complex symptom definitions. In some cases, these complex symptom definitions are not yet deployed in the production environment. Problem-management tools can be launched in context to consult offline symptom catalogs that contain newly created symptom definitions that have been validated in a test or preproduction environment. Access to this expanded knowledge base might identify symptoms that were not previously identified in the manual incident-management process. A small set of matching probable-cause symptoms can be rapidly and accurately identified. Based on historical data, one of these symptoms is selected as the most likely root cause of the problem. A problem record is created with the information contained in the root-cause symptom. In our example, the root-cause symptom indicates that a memory leak situation has been identified.

Root causes of unknown errors can be identified more rapidly by using development tools that use the common event format and symptom knowledge described in the ACRA. Advantages when an automated symptom analysis and processing strategy is used include access to a larger knowledge base, thus providing a more accurate diagnosis, and the possibility of executing automated corrective actions as a result of the symptom-analysis process.

Resolve problem and record resolution

In the current process, the problem recovery RFC (apply software patch) is carried out after approval by the change manager, and the newly detected and diagnosed problem is recorded as a known error, often by problem-resolution specialist staff. Problem and associated incident records are closed.

For the autonomic process, the corrective RFC (apply software patch) is extracted from the root-cause symptom definition and submitted for execution. The application of the patch involves the change management process (described in the next case study), and that process might also involve some form of automation. After the RFC is processed, event-management monitoring indicates that the server is correctly functioning again. Events, incidents, and problems are closed. In addition, the new symptom definition that identified the problem is migrated to the production-level knowledge base as a new known error so that future occurrences of related incidents can be handled by incident management (this constitutes proactive problem management). Common-problem remedial-action recording, as described in the ACRA and embodied in symptom definitions, enables automated resolution of problems. Moreover, this same capability enables unknown problems that are detected during operation, along with their associated resolution, to be migrated to known errors that can be handled automatically in the future.

ITIL SOLUTION DEPLOYMENT

The ITIL solution deployment case study shows how autonomic concepts and technologies may be used in the context of the ITUP solution-deployment processes. The configuration-management process is used to identify, record, and report IT components (hardware, software, and documentation), including their versions, constituent components, and relationships. The controlled assessment and approval of infrastructure changes is provided by the change-management process. The controlled introduction of collections of related authorized changes is accomplished by the release-management process.

This case study shows that when ITSM solutions are designed and integrated, the delegation and partial-autonomic-manager architectural patterns enable autonomic capabilities to be composed in an incremental and flexible way. It also demonstrates the hierarchical-autonomic-manager architectural pattern and accommodation of organizational boundaries.

In addition to demonstrating the advantages of these patterns, this case study shows the importance of using the SDD (Solution Deployment Descriptor) as the canonical representation of both solution deployment and dependency knowledge. That knowledge is then used in the analysis, planning, and execution of key tasks within the solution-deployment processes. These tasks include dependency and impact analysis, configuration drift analysis, and release planning.

This case study is described with two scenarios. As shown in *Figure 6*, the overall flow for scenario 1 starts when a release builder constructs a software package and its corresponding SDD. This is stored in the definitive software library (DSL) by the configuration librarian and is registered in the CMDB, making it available for deployment. A change analyst who processes an RFC for a new software release employs an assessment tool to analyze the impact of the change, using information in the CMDB. A release planner then processes the RFC, using a planning tool to plan the rollout of the release.

Scenario 2 handles the rollout of the release. The rollout plan passes to a release implementor, who uses an orchestration tool that coordinates the deployment by using one or more local installation tools to install the software on the target resources.

The details of these scenarios are described in the following sections. As in the other case study, each activity in a scenario is described by contrasting the current process with the process that has been enhanced with autonomic capabilities (the autonomic process).

Solution-deployment scenario 1—Software-release management

In solution-deployment scenario 1, we investigate the use of autonomic technology to support the

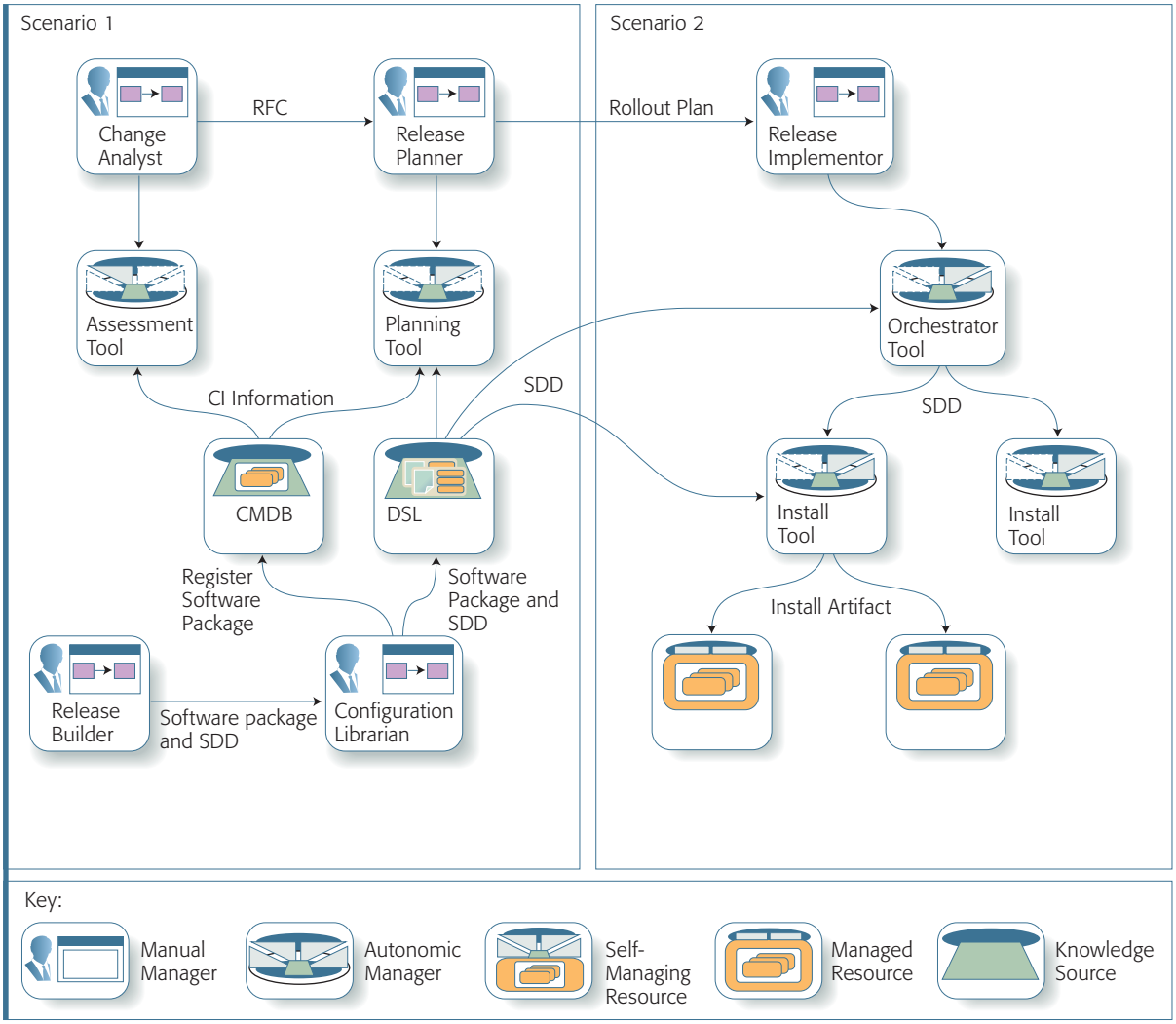

Figure 6
Autonomic building blocks for release management scenarios

development and planning for a new software release, which consists of a human resource (HR) application and an upgrade to the Web application platform on which it is deployed.

The main application code for the HR application will be deployed in a J2EE** server. The first release of the HR application is to be deployed in three locations. Two of those are completely new deployments. The third location has a previous version of the standard Web platform, which is running two existing applications. The two new locations will share one J2EE server, and the third will continue to have its own dedicated J2EE server and Web server.

RFCs are raised and tentatively assigned to a release

Using the current process, three RFCs are raised for the three locations where the application is to be deployed. Each RFC requires different approvals, but all RFCs are an intended part of a single release.

In the autonomic process, the development team associates an SDD for the HR application with the RFCs to describe the required changes. This provides a "contract" between development and operations that describes the dependencies that the HR application has on its environment. The RFC identifies the intended target environments in each location to satisfy those dependencies.

RFCs are assessed for impact

In this activity for the current process, a change analyst works with subject-matter experts to assess the RFCs for their likely impact. The ability to do this effectively is limited by the lack of accurate information about the dependencies of the software being installed and the dependencies of existing software that could be impacted by the release. In this scenario, the new installations are assessed as a minor impact because they are to be installed on new, dedicated servers with firewall software. The third installation is assessed as a significant impact because it requires upgrading servers that host existing applications. Significant manual investigation is needed to determine that the updated version of the J2EE server is compatible with one of the existing applications, but that the other will require an upgrade. An additional RFC is raised to address this, and this new RFC is linked to the RFCs that depend on it.

In the autonomic process, the change analyst uses an automated impact assessment tool in automated-assistance mode to verify that the intended target environments meet the dependencies described in the SDD and to ensure that the potentially impacted applications have been correctly identified in the RFC. As shown in Figure 6, the assessment tool is a partial autonomic manager that uses standard WSDM interfaces to access information in the CMDB and DSL. The automation of this analysis relies on a consistent definition of resource types used by both the SDD authors to specify requirements and the CMDB to describe resource instances.

By analyzing the SDDs for the new and existing applications, and comparing them to information in the CMDB, the tool is able to determine automatically that the updated version of the J2EE server is compatible with one of the existing applications but that the other will require an upgrade. An additional RFC is automatically generated based on the dependency information in the SDD. The tool also identifies and generates RFCs for the required operating-system and database-driver updates.

The HR application SDD also identifies the need for specific security settings on the Web server. Because this level of detail is not stored in the CMDB, the tool reports the need to investigate the settings. The change analyst forwards details to the application-server support team, who confirm that a configuration change and recycle will be needed and who provide configuration scripts to perform this change.

Changes are approved and scheduled

In the current process, the RFCs in the release are assessed by the change advisory board, approved, and scheduled for the first maintenance window following planned completion of the release cycle. During scheduling, it is determined that there is a minor operating-system update scheduled for a week before the application release. The change analyst has to consult with the appropriate specialists to ascertain that the J2EE server supports the new operating-system level, and the change analyst sends a note to the release team that testing should be performed on the new operating-system level. The approved RFCs are passed to release management for implementation.

In the autonomic process, the scheduling takes into account the need to recycle the Web server and the additional prerequisite RFCs by using the additional information obtained during the change assessment activity in the previous autonomic process. Because of the detailed dependency information, it is also possible to take into account information about pending changes. The change analyst is notified about an operating-system update that will occur before the release. Automated dependency analysis shows that the new operating-system level is supported by the J2EE server.

Release is built and tested

In the current process, the release specialists assemble and test all of the required content and create scripts for the build, installation, and rollback of the release. At this point, missing or inaccurate dependencies may be discovered, requiring modification of the RFCs and reapproval. The release specialists must communicate closely with the development team and with operational teams, such as security specialists, to ensure that the application is refined appropriately for operational deployment, configured to run on the standard Web platform, and meets current security policies.

In the HR application scenario, release testing uncovers that some operating-system and database-driver updates are required. These changes are raised and tagged as prerequisites that must be applied in the same maintenance window. It is also discovered that the security settings of the Web

server need to be updated to support the application, and that for these settings to take effect, the Web server must be recycled. This requires an additional RFC to be approved and postponement of some lower-priority changes scheduled for the same maintenance window to allow time for the recycling. The authorized and tested application package is stored in the DSL and registered in the CMDB.

When using the autonomic process, the release builder constructs a solution package composed of the HR application SDD, configuration scripts, and other software that is part of the release, including prerequisite updates and fixes. This SDD is used to deploy the acceptance version of the release. Once acceptance testing is complete, the new software package and SDD are stored in the DSL and registered in the CMDB. The manual effort to deal with dependencies is eliminated or greatly reduced.

Detailed rollout plan is created

Using the current process, the release planner creates a plan to install the release, taking into account the dependencies among elements of the release and their installation procedures (for example, the need to recycle servers for changes to take effect). The execution of this rollout plan is described in scenario 2.

With the autonomic process, the release builder provides the necessary instance parameters for each location to instantiate the solution SDD on the specific targets with appropriate configuration values. The combination of the instance parameters and the solution SDD constitutes the rollout plan for the software aspects of the release. The manual effort to create a plan is greatly reduced because the SDD contains sufficient information to automate the deployment of the software and its dependencies.

Solution-deployment scenario 2—Automated orchestration

In solution-deployment scenario 2, we examine the role of autonomic technology in deploying the release that was planned and built in scenario 1.

Once approved and scheduled, a release implementor is assigned to coordinate the release deployment. The release implementor may be empowered to make all of the necessary changes or may need to work with specialists in different areas of the organization.

The release implementor works with specialists to implement the rollout plan defined in scenario 1. This is likely to be error-prone, particularly when the rollout plan is merely documented rather than being captured in automated scripts. In cases in which automated scripts are defined, they might be environment-specific and might not accommodate the variability in the actual deployment locations.

In the autonomic process, we assume that the release implementor is empowered to perform all of the work and uses automated deployment tools to implement the release. As shown in Figure 6, this consists of an orchestration tool (a "planning" autonomic manager) that is capable of generating orchestration flows from SDDs. This recent enhancement is still under evaluation, but based on testing to date, the capability is trusted sufficiently to be used in production, although the release implementor is required to verify the plan that the tool creates. The autonomic manager is therefore used in "supervised delegation" mode.

This sequence of activities is followed during deployment:

1. For each location, the release implementor retrieves the rollout plan that identifies the solution package and the instance parameters for the deployment.
2. The deployment orchestration tool is used to map the release plan to the actual physical nodes used for the deployment, based on the data model of the tool.
3. The deployment orchestration tool retrieves the relevant SDDs from the DSL and checks the SDD requirements against the known configuration of existing resources from the CMDB or from a local data model (queried using WSDM interfaces). It informs the release implementor of any issues.
4. The deployment orchestration tool uses the SDDs to generate an orchestration plan based on the physical targeting information and parameterization information in the release plan.
5. The release implementor reviews the plan and approves it.
6. The orchestration tool orchestrates the distribution of software (retrieved from the DSL) and its subsequent deployment. Once the software has been distributed to an endpoint, the orchestration tool invokes a local installation tool at that

endpoint (an implementation of the hierarchical-autonomic-manager pattern).
7. Each local installation tool performs dependency checking, planning, and execution for the locally targeted software, using WSDM interfaces.
8. When deployment is complete, the CMDB is updated to include the newly deployed or updated resources as part of the authorized baseline.

The use of the SDD and autonomic-manager capabilities for release implementation allows the release implementor to deploy a release that is more reliable and repeatable and has less likelihood of human error or miscommunication. It also reduces the need to involve subject-matter experts in the deployment. Automated planning and execution can reduce the time to execute and provide both a clear audit of actions taken and the ability to perform automated rollback. Updating the CMDB-approved baseline with SDD knowledge reduces misidentification of configuration audit issues and improves the accuracy of CMDB data.

CONCLUSION

Since the inception of the autonomic-computing initiative, the growth and maturation of autonomic capability has been predicated on a phased, progressive model designed to deliver demonstrable business value. As shown here, delivery of the requisite autonomic capabilities is predicated on key technologies, specifications, and architectural patterns within the broader context of the ACRA.

As we have demonstrated throughout the ITIL-based scenarios presented here, the value of autonomic computing is manifest in its ability to effect incremental IT process transformation. More important, autonomic computing, as described by its reference architecture and associated elements, delivers a framework for progressive, business-driven evolution of IT infrastructures and associated processes.

ACKNOWLEDGMENTS

The authors acknowledge our IBM colleagues who have contributed to the evolution of the concepts described in this paper. In particular, we thank the following people: John Sweitzer, Dave Ehnebuske, Randy George, Jim Crosskey, Pat Rago, Hari Madduri, David Kaminsky, Jim Whitmore, Mark Weitzel, La Tondra Murray, and Ric Telford.

CITED REFERENCES

1. *Autonomic Computing: IBM's Perspective on the State of Information Technology*, IBM Corporation, http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.
2. A. G. Ganek and T. A. Corbi, "The Dawning of the Autonomic Computing Era," *IBM Systems Journal* **42**, No. 1, 5–18 (2003).
3. *IBM Paves the Way for Mainstream Adoption of Autonomic Computing*, IBM Corporation, http://www-03.ibm.com/autonomic/press_041905.shtml.
4. Dynamic Systems Initiative, Microsoft Corporation, http://www.microsoft.com/business/dsi/default.mspx.
5. Adaptive Enterprise, Hewlett-Packard Development Company, L.P., http://h71028.www7.hp.com/enterprise/cache/6842-0-0-225-121.html.
6. Sun N1 Grid Engine 6, Sun Microsystems Inc., http://www.sun.com/software/gridware/.
7. Autonomic Computing, IBM Research, http://www.research.ibm.com/autonomic/research/.
8. The 3rd International Conference on Autonomic and Autonomous Systems, Athens, Greece (June 2007), http://www.iaria.org/conferences2007/ICAS07.html.
9. The 4th IEEE International Conference on Autonomic Computing, Jacksonville, FL (June 2007), http://www.acis.ufl.edu/~icac2007/index.shtm.
10. International Conference on Self-Organization and Autonomous Systems in Computing and Communications, Erfurt, Germany (September 2006), http://www.soas2006.org/.
11. J. Kephart and J. O. Strassner, "Autonomic Systems and Networks: Theory and Practice," *Proceedings of the IEEE/IFIP Network Operations and Management Symposium*, Vancouver, Canada (2006), p. 588.
12. R. Cutlip and C. Zabeu, *Autonomic Computing: Strengthening Manageability for SOA Implementations*, White Paper, IBM Corporation (December 2006), http://www-03.ibm.com/autonomic/pdfs/AC_SOA_EB.pdf.
13. C. Draper, *Combine Autonomic Computing and SOA to Improve IT Management*, IBM Corporation (April 2006), http://www-128.ibm.com/developerworks/library/ac-mgmtsoa/index.html.
14. R. Cutlip, "SOA and Autonomic Computing," *Proceedings of The Open Group: Enterprise Architecture Practitioners Conference*, San Diego, CA (January 2007).
15. Autonomic Computing Industry Standards, IBM Corporation, http://www-03.ibm.com/autonomic/industry.html.
16. V. Tewari and M. Milenkovic, "Standards for Autonomic Computing," *Intel Technology Journal* **10**, No. 4, 275–285 (2006).
17. Information Technology Infrastructure Library (ITIL), Office of Government Commerce, http://www.itil.org.uk.

18. K. Behr, G. Castner, and G. Kim, "The Value, Effectiveness, Efficiency, and Security of IT Controls: An Empirical Analysis," http://www.itpi.org/docs/veesc.pdf.
19. An Architectural Blueprint for Autonomic Computing, White Paper, IBM Corporation (June 2006), http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf
20. OASIS Web Services Distributed Management, Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm.
21. Common Information Model (CIM) Standards, Distributed Management Task Force, Inc., http://www.dmtf.org/standards/cim/.
22. D. Kaminsky, *An Introduction to Policy for Autonomic Computing*, IBM Corporation, http://www-128.ibm.com/developerworks/autonomic/library/ac-policy.html.
23. *OASIS Standard wsdm-muws-part1-1.0*, "Web Services Distributed Management: Management Using Web Services," W. Vambenepe, Editor, Organization for the Advancement of Structured Information Standards (March 9, 2005), http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part1-1.0.pdf.
24. H. Kreger, *A Little Wisdom About WSDM*, IBM Corporation, http://www-128.ibm.com/developerworks/webservices/library/ws-wisdom/.
25. D. Ogle, H. Kreger, A. Salahshour, J. Cornpropst, E. Labadie, M. Chessell, B. Horn, J. Gerken, J. Schoech, and M. Wamboldt, *Canonical Situation Data Format: The Common Base Event V1.0.1*, IBM Corporation, http://www.eclipse.org/tptp/platform/documents/resources/cbe101spec/CommonBaseEvent_SituationData_V1.0.1.pdf.
26. IBM Autonomic Computing Symptom Specification, Symptoms Reference Specification, Version 2.0, IBM Corporation, http://download.boulder.ibm.com/ibmdl/pub/software/dw/opensource/btm/SymptomSpec_v2.0.pdf.
27. M. Perazolo, *Symptoms Deep Dive, Part 1: The Autonomic Computing Symptoms Format*, IBM Corporation, http://www-128.ibm.com/developerworks/autonomic/library/ac-symptom1/.
28. OASIS Solution Deployment Description (SDD) Technical Committee, Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=sdd.
29. IBM Tivoli Unified Process, IBM Corporation, http://www-306.ibm.com/software/tivoli/governance/servicemanagement/itup/tool.html.

Accepted for publication January 2, 2007.
Published online June 27, 2007.

Peter Brittenham
IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (peterbr@us.ibm.com).
Mr. Brittenham is a Senior Technical Staff Member within the autonomic computing organization. Mr. Brittenham is currently the chief programmer for autonomic computing focused on delegation and self-managing resources to enable IT service management. He has a B.S. degree in business administration from Boston University and an M.S. degree in computer science from Marist College.

R. Russell Cutlip
IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (cutlip@us.ibm.com).
Mr. Cutlip is a software architect on the IBM autonomic-computing architecture team. His current interests include the interplay of autonomics business processes and SOA. Mr. Cutlip is coauthor of *Integrated Solutions with DB2*, published by Addison-Wesley in 2003. He has an M.B.A. degree in management of information systems and strategic planning from the University of Pittsburgh.

Christine Draper
IBM Software Group, 11501 Burnet Road, Austin, Texas 78758 (cdraper@us.ibm.com).
Mrs. Draper is a Senior Technical Staff Member on the IBM autonomic-computing architecture team and works with development teams within IBM to apply autonomic computing concepts to IT service management. An active member of OASIS, she is working to standardize a solution deployment descriptor. She has a B.A. degree with honors in natural sciences from the University of Cambridge, England.

Brent A. Miller
IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (bamiller@us.ibm.com).
Mr. Miller, Senior Technical Staff Member on the IBM autonomic-computing architecture team, is the lead architect in autonomic computing, addressing both technology and standards efforts associated with self-managing autonomic systems and ITSM. He has a B.S. degree in computer and information science from the Ohio State University. He is coauthor of *Bluetooth Revealed*, published by Prentice-Hall in 2000.

Samar Choudhary
IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (samar@us.ibm.com).
Dr. Choudhary is currently the chief architect for the integrated-solutions console project at IBM. Dr. Choudhary has an M.S. degree in computer science from the University of Minnesota and a Ph.D. degree in mathematics from Northern Illinois University.

Marcelo Perazolo
IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (mperazol@us.ibm.com).
Mr. Perazolo is a member of the IBM autonomic-computing architecture team, where he serves as senior architect for symptoms, virtualization, and knowledge representation. He has B.S. and M.S. degrees in electrical engineering, both from the State University of Campinas, Brazil. ■