under new

Autonomic computing is revolutionizing the way we manage complex systems.

In an increasingly competitive global environment, enterprises are under extreme pressure to reduce operating costs. At the same time they must have the agility to respond to business opportunities offered by volatile markets.

Leveraging IT for competitive advantage is fundamental to a company's success, while maximizing its use of IT infrastructure is crucial to cost reduction. Moreover, architecting systems to handle peak activity and to provide adequate business continuity backup is a prerequisite that adds to both capital and operating costs.

Implementing a shared IT infrastructure, or *service grid*, where IT resources can be made available, dynamically, to satisfy the needs of different business units, is a compelling business proposition for CIOs. Actually implementing such an architecture is challenging, however, and often the management overhead associated with its operation cancels out any capital cost savings.

One solution is to adopt what we call an efficient computing architecture, combining the necessary provisioning, virtualization, and automation components required for the 24/7 operation of such a service grid.

management

DUNCAN JOHNSTON-WATT, ENIGMATEC

In this article we focus on the automation component that manages applications running in a service grid. This component is known generically as an EMS (execution management system). It enables business-oriented requirements such as service-level objectives to be mapped onto the operational processes or strategies that orchestrate IT resources at the hardware, operating system, and application stack levels.

We examine the fundamentals of an EMS implementation we have built using a POM (process-oriented middleware) platform and its relationship to autonomic

computing. POM is the next generation beyond MOM (message-oriented middleware). MOM virtualizes data distribution, whereas POM virtualizes process execution, optimizing the execution of business or, in this case, operational processes in a highly distributed computing environment.

THE SERVICE GRID

The principal challenge for CIOs of global enterprises today is to maximize the use of their IT resources while supporting an environment that is under constant

under new management

change and subject to increasing regulatory scrutiny. They are charged with reducing the total cost of ownership of their IT infrastructure—a business imperative in a global world of increasing competition and decreasing trading margins—against a backdrop of increased market volatility, rapidly changing business opportunities and customer requirements, and where operational resilience is of fundamental importance.

The requirements for implementing an enterprise-wide IT platform can be specified by these characteristics:

- **Utilization**—maximizing the usage of IT resources, including servers, storage, and network bandwidth, while acknowledging the necessity to provide appropriate operational resiliency and separation (the parameters of which may be specified and mandated by industry regulators).
- **Agility**—repurposing IT resources dynamically in order to fulfill the changing needs of multiple lines of business operating across multiple time zones, paying due attention to the company's overall business priorities.
- **Scalability**—dynamically provisioning IT resources to meet the changing requirements of the company's business as a whole, ensuring that appropriate IT resources are available but not wasted.

Traditionally, individual lines of business have taken ownership of their own IT platforms, ensuring that resources are available to meet peak workloads and support failover conditions. Although this approach satisfies the need for IT resources at any time, it is inefficient in terms of utilization and lacks flexibility and scalability at an enterprise level. In practice, IT resources often remain idle or barely utilized, at significant cost to the company.

To achieve a cost-effective approach to the provisioning of IT resources, companies are looking to build shared infrastructures, leveraging industry-standard hardware components (such as blade servers and symmetric multiprocessing clusters) capable of supporting multiple heterogeneous operating systems (e.g., Linux, Solaris, Microsoft Windows) and application environments.

These shared infrastructures, or service grids, are dynamically partitioned and allocated to meet the needs of multiple lines of business as their requirements for IT resources change. IT components within the service grid are powered up and down and repurposed as required by the application mix at any given time.

For this service grid architecture to be feasible, however, management function is required to arbitrate on resource conflicts and to determine priorities across lines of business and their application environments. Moreover, these priorities cannot be viewed as static parameters and are likely to change according to business needs and other factors such as time of day.

Underpinning this management function is a set of requirements, or SLAs (service-level agreements), that specify the IT resources needed by a line of business and the specific applications that they maintain. The service grid management function consults these SLAs and maps them to a set of priorities mandated by the business, at an enterprise level, to determine which resources to allocate at any time on a best-efforts basis.

THE EFFICIENT COMPUTING ARCHITECTURE

The service grid management function optimizes the utilization of IT resources on a dynamic basis, without the need for operator intervention, by implementing service-level automation. In this context, the role of an EMS is to deliver this service-level automation underpinned by policy-driven management. EMS can be thought of as the glue that binds all the components found in a typical enterprise IT infrastructure:

FIG 1

- **Hardware**—servers, storage, and networks
- **Virtualization**—server, storage, and network virtualization tools
- **Monitoring**—network and systems monitoring tools
- **Provisioning**—operating system and software installation tools
- **Infrastructure software**—application servers, databases, middleware
- **Enterprise applications**—third-party and bespoke applications

This creates the overall efficient computing architecture, as illustrated in figure 1.¹

From the perspective of business managers, an EMS is typically configured to manage resources at the application level, the line-of-business level, and the service-grid level. At each level, requirements for IT resources are notified to the next level in the hierarchy.

To deliver an enterprise-class closed-loop solution, an EMS needs to integrate with all the components shown in figure 2. To instrument and manage these efficiently, an EMS needs to be distributed. Our approach is to achieve this using a POM platform.

The policies that define the management of some aspect of the IT infrastructure can be modeled as a set of workflows or operational processes triggered by various scenarios, where an operational process is a business process in the operations domain. A POM platform virtualizes process execution, optimizing the runtime execution of these processes in a highly distributed computing environment. It achieves this by analyzing these processes and decomposing them into their constituent

parts, which are then deployed and constrained to run close to the resources under management.

AUTONOMIC COMPUTING

The concept of autonomic computing, as envisioned by IBM,² is a deliberate reference to the biological *autonomic nervous system* that automatically regulates a living being's heart rate, body temperature, and other core functions. The goal of autonomic computing is to emulate this system by creating self-governing computer systems that can configure, heal, optimize, and protect themselves without human intervention.

An autonomic computing system consists of autonomic elements. These are logical constructs that monitor some aspect of a system, analyzing its output and taking specific actions to adjust it, so that the overall system is able to meet specific business objectives, typically expressed as SLAs.

Autonomic elements are self-organizing. They can discover each other, operate independently, negotiate or collaborate as required, and organize themselves so that the emergent stratified management of the system as a whole reflects both the bottom-up demand for resources and the top-down business-directed application of those resources to achieve specific goals.

EMS AND AUTONOMIC COMPUTING

An EMS built using a process-oriented middleware platform can be viewed as an autonomic computing system. Leveraging such a platform enables us to deconstruct the complex problem of service-grid management into its constituent parts by creating and deploying self-organizing, fully redundant EMS agents, where each agent is an autonomic element.

In their landmark paper, *The Vision of Autonomic Computing*, Jeffrey Kephart and David Chess observed that “viewing autonomic elements as agents and autonomic systems as multi-agent systems makes it clear that agent-oriented architectural concepts will be critically important.”³ A process-oriented EMS recognizes and exploits this duality between agents and autonomic elements, using autonomic policy-driven management to deliver service-level automation.

This approach has significant advantages over the traditional “mission-control” approach to IT management where events are correlated centrally and actions initiated manually. Not only does it minimize network traffic, but it also ensures that in a disaster recovery scenario the EMS will continue to operate since it is a distributed peer-to-peer system.

under new management

In fact, if there is a network outage and one part of the IT infrastructure such as a data center becomes isolated, a process-oriented EMS will automatically split into two instances, each capable of dealing with the outage correctly from its point of view. For example, the EMS instance associated with the isolated data center can initiate a policy to gracefully shut down any production services it is responsible for, knowing that its counterpart will be starting up these services on backup resources elsewhere in the infrastructure.

EMS AGENTS ARE AUTONOMIC ELEMENTS

As shown in figure 2, an autonomic element consists of one or more managed elements together with an autonomic manager that governs their behavior according to some rules or policies. A managed element must support both a sensor and an effector interface. The sensor interface specifies the metrics that the managed element can emit, which the autonomic manager can monitor. The effector interface specifies the operations that the autonomic manager can invoke to change the behavior of the managed element.

In a process-oriented EMS implementation, a managed element can be either an EMS agent or an EMS adapter that connects to some component in the IT infrastructure. Sometimes referred to as BMEs (base-managed elements), these integration points with IT resources are very similar to the notion of a concrete managed element in the common information model.

In classic autonomic computing, the autonomic manager is responsible for implementing a control loop, often referred to as a closed loop, that reacts to events from managed elements and may, as a result, change or affect their behavior (see figure 3).

The autonomic manager consists of four main functions, known as MAPE (monitoring, analyzing, planning, and execution).

- **Monitoring** is the collection, aggregation, and filtering of information collected from one or more managed elements.
- **Analyzing** is the correlation of this information and the modeling of complex situations that allow the autonomic manager to learn about its environment and predict future situations.

- **Planning** is the structuring of those actions needed to achieve specific policy-driven goals and objectives.
- **Execution** is both the realization of the resultant plan and the management and application of dynamic updates to the plan.

An autonomic element represents a logical unit and can present its own managed element interface and be subject to autonomic management by another autonomic manager. For example, imagine an autonomic manager responsible for controlling the temperature of a system blade (TempControlME). This would have associated with it several BMEs for detecting the temperatures (TempBMEs) and controlling the temperature (FanBME). The TempControlME would report its relative success or failure at controlling the temperature and provide a means to regulate the temperature at varying levels based upon other conditions (see figure 4).

Besides using its sensor interface as a reporting mechanism (e.g., by emitting a synthetic metric), an autonomic element may detect a situation with which it is not familiar. It then has the option to escalate this via its sensor interface.

Another common scenario is where an autonomic manager does not have control of or access to specific resources it requires to complete a plan. It may need to escalate a request for more resources, identifying itself in such a way that it can be notified via its effector interface when these resources become available. Conversely, the autonomic element can advertise its services and act as a delegate for another autonomic element (see figure 5).

In a process-oriented EMS, an EMS service is the direct analog of the autonomic manager. It is constructed from a set of associated managed elements and a set of policies encoded as workflows. It also presents its own sensor and effector methods.

An EMS service implements the standard MAPE ele-

ments that make up an autonomic manager:

- **Event translation (monitor).** Since an EMS is fundamentally an event-driven system, events are either derived from EMS agents (sometimes referred to as managed EMS services) or EMS adapters, which capture external events and translate them into our canonical internal messaging format.

- **Situation analyzer (analyze).** A situation is a sequence of events that matches an event pattern. An EMS situation analyzer is a mechanism for simplifying the process of describing these complex situations. When such a complex situation is detected, a condition may be evaluated to restrict the situation further. An EMS situation analyzer acts as a message preprocessor for an EMS service. As an EMS scenario can be started only by a single message, the EMS situation analyzer lets you distill the input from a number of messages down to a single message. Only one EMS situation analyzer per scenario is allowed.

- **Scenario (plan).** An EMS scenario diagram works within the limits set by a constraint diagram, which describes the set of sensors and effectors associated with an EMS service. An EMS scenario diagram is a workflow that describes the operational procedure that makes controlled change to particular managed elements, via their effectors, based on an initial triggering event-condition or situation-condition.

- **Scenario invocation (execute).** The runtime execution of a given scenario occurs when its associated event-condition or situation-condition is detected. In many cases this will result in an asynchronous distributed execution of a complex workflow that interacts with multiple managed elements associated with a particular EMS service.

- **Data space (knowledge).** To detect trends and maintain continuity of policy, a means of storing and retrieving data in a distributed environment is required. This is necessary to be able to provide shared state in a distributed environment.

MANAGING A SERVICE GRID

A process-oriented EMS sees the management space as an organized hierarchy of managed elements and autonomic managers that mirrors the way an IT organization is structured, so the notions of escalation and delegation are natural ways of modeling the interactions.

For example, in an investment bank, EMS application agents manage each institutional equities application and publish their resource requirements to the EMS business agent managing this business unit. Likewise, EMS application agents managing credit derivatives and fixed-income applications notify their respective EMS business agents

under new management

of their resource requirements (see figure 6).

Both EMS application agents and EMS business agents can be configured to maintain some local headroom, but the resources are ultimately owned by the EMS service-grid agent. This agent leases resources to the EMS business agents, which in turn can lease or delegate them to individual EMS application agents.

The primary function of both the service-grid agent and the business agent is to enforce business SLAs by prioritizing resource allocations and resolving resource conflicts and, where necessary, by instructing EMS agents in their domain to take action (e.g., yield resources if a higher-priority need is identified and there isn't sufficient spare capacity to service this request in the short term).

The primary function of the individual EMS application agents is to manage the scale-in and scale-out of their individual application stack and compute the resources required to operate their application or service within acceptable technical SLAs.

EMS SERVICE GRID AGENT

The EMS service-grid agent supports a standard sensor/effector interface that facilitates its integration with standard network and systems monitoring tools or EMS agents responsible for higher-order policies such as implementing the company's business continuity/disaster recovery strategy. It has four key areas of responsibility:

Server pool management. The EMS service-grid agent owns and tracks servers and maintains its server pool. It recognizes new servers when they are brought online and adds these to the pool. It handles predictive failure and routine maintenance, de-allocating servers where appropriate, taking them offline and removing them from the pool.

Line-of-business management. The EMS service-grid agent owns the life cycle of each of its EMS business agents (e.g., it tracks the status of each line of business and if it cannot meet its minimum operational requirements, it will suspend its operations).

Resource allocation. The EMS service-grid agent acts as a resource broker. It handles EMS business agent resource updates or hints, which are expressed as minimum, current, and ideal resource requirements; and it determines the actual resource allocations (e.g., if an EMS business agent increases its ideal requirement, then the EMS service-grid agent attempts to match this by leasing the required resources from the service-grid server pool, thus ensuring that the EMS business agent maintains its preferred headroom).

Resource contention. In the unlikely event that the EMS service-grid agent cannot satisfy a specific EMS business agent resource request in full because of exceptional demand on the service grid, it will automatically apply policies to resolve this resource conflict in accordance with the SLAs accepted by the business managers.

RESOURCE CONTENTION ALGORITHM

The EMS service-grid agent first determines the subset of existing resource allocations that are lower priority. Then it iterates over the EMS business agents responsible for managing these lower-priority resource allocations until they've yielded sufficient resources to satisfy the request.

This algorithm can be tuned to operate breadth first (in which case it tries to maintain the minimum allocation

for all lower-priority business units) or depth first (in which case it reclaims all resources required from lower-priority business units, starting with the lowest priority and potentially resulting in suspension of service).

In any suboptimal situation—when adequate resources are not available—the goal of the EMS service-grid agent is to try to provide the EMS business agent with some headroom, by ensuring that the actual allocation exceeds the last known current requirement held in shared state. Failing this, it tries to ensure that at the very least the minimum requirement is met; otherwise, it instructs the EMS business agent to suspend its line of business until adequate resources become available.

EMS BUSINESS AGENT

The EMS business agent is responsible for managing resources at the line-of-business level. It has five key areas of responsibility:

Resource management. The EMS business agent tracks the resource requirements of individual applications within its line of business. It tries to maintain adequate headroom to respond to *reactive* requests for more resources from these applications, but its primary goal is to preempt these requests by providing its applications with adequate headroom at all times. This *predictive* allocation of resources in advance is achieved by applying rules that are derived from analyzing historical usage patterns and understanding both intraday usage patterns and the impact of external events (e.g., U.S. Treasury announcements).

Resource allocation. The EMS business agent maintains a local server pool of resources leased to it by the EMS service-grid agent. It supports the standard operations:

- **allocated(resourceID).** It adds compute resources leased to line of business to its local pool.
- **yield(resourceID).** Since all its resources are leased by the EMS service-grid agent, it is obliged to free up any resource—or its logical equivalent—when required.

Note that since yielding a resource can take some time, the EMS business agent handles this asynchronously. Therefore, upon completion of this request, a **yielded(resourceID)** message is sent to the EMS service-grid agent. As a good citizen the EMS business agent is also expected to free up resources it no longer requires, effectively returning them to the service-grid server pool. It indicates that it has done so by sending an **unsolicited released(resourceID)** message to the EMS service-grid agent.

Resource requirements. The EMS business agent is responsible for providing the EMS service-grid agent with

under new management

the following key metrics:

- Minimum compute requirements—indicates resources (CPUs, granularity) required to deliver minimal service.
- Current compute requirements—indicates resources (CPUs) actually in use.
- Ideal compute requirements—indicates resources (CPUs, granularity) required to deliver its service to SLA.

Resource contention. In the unlikely event that the EMS business agent cannot satisfy a specific EMS application agent resource request in full, because of exceptional demand on the service grid, it will automatically apply policies to resolve this resource conflict in accordance with the SLAs, using fine-grained variation on the resource contention algorithm previously described.

Standard life-cycle management. The EMS business agent supports standard life-cycle operations `start()`, `suspend()`, `resume()`, and `stop()`. It also reports its status. This interface enables the EMS service-grid agent to own and manage the life cycle of the line of business.

EMS APPLICATION AGENT

In a complex multitier application, the EMS application will follow a similar pattern to the EMS business agent and assume responsibility for managing resources at the local application level, maintaining local headroom, and so on. This fractal infrastructure management pattern is a feature of autonomic systems such as EMS.

The EMS application agent also implements the appropriate scale-out and scale-back policies to maintain its technical/performance SLAs. There are many standard infrastructure design patterns, such as compute grid (master/worker) and application grid (logical partitioning).

SEMANTIC WORKLOAD MANAGEMENT

A process-oriented EMS makes it straightforward to implement strategies such as policy-based semantic workload management. SLAs are used to ensure that critical applications are distributed across the service grid in a fashion that draws on business requirements, as well as technical resource optimization.

AN EFFICIENT COMPUTING SOLUTION

A process-oriented EMS should combine policy design with policy execution to automate all aspects of data-

center operations, providing a powerful solution to the increased demands on today's complex data center. It should be capable of replacing error-prone scripted and manual procedures and can automate systems at any level, from shutting down a server gracefully to moving an entire suite of applications to a disaster recovery site.

It should provide a comprehensive design-time workbench that facilitates the capture, testing, and maintenance of operational workflows as powerful, reusable policies. It should provide a runtime environment that is capable of detecting failure or changes in demand instantly, and that coordinates the automatic execution of the policies required to maintain system performance, while optimizing utilization and significantly lowering operational risk.

The integration of an EMS with third-party middleware, monitoring, provisioning, and virtualization technologies—or integrated service grid management—delivers an efficient computing solution. □

REFERENCES

1. Gentzsch, W., Iwano, K., Johnston-Watt, D., Minhas, M. A., and Yousif, M. 2005. Self-adaptable autonomic computing systems: An industry view. In *Proceedings of the 3rd International Workshop on Self-Adaptive and Autonomic Computing Systems*, Copenhagen, Denmark (August 24-25).
2. IBM Autonomic Computing Research; <http://www.research.ibm.com/autonomic/>.
3. Kephart, J., and Chess, D. 2003. The vision of autonomic computing. *IEEE Computer* 36(1): 41-50.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

DUNCAN JOHNSTON-WATT is the principal founder of Enigmattec Corporation (<http://www.enigmattec.net>). He has more than 15 years of experience developing technology specializing in the development of large-scale systems. An early adopter of Java enterprise technologies in the financial services industry, he was nominated for a *Computerworld* Smithsonian Award in April 2000. Duncan holds an M.Sc. in computation from Oxford University.

© 2006 ACM 1542-7730/06/0300 \$5.00