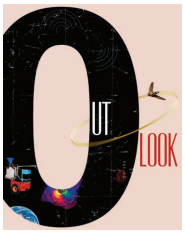
The Vision of Autonomic Computing



Systems manage themselves according to an administrator's goals. New components integrate as effortlessly as a new cell establishes itself in the human body. These ideas are not science fiction, but elements of the grand challenge to create self-managing computing systems.

Jeffrey O. Kephart
David M. Chess
IBM Thomas J. Watson Research Center

In mid-October 2001, IBM released a manifesto observing that the main obstacle to further progress in the IT industry is a looming software complexity crisis.[1] The company cited applications and environments that weigh in at tens of millions of lines of code and require skilled IT professionals to install, configure, tune, and maintain.

The manifesto pointed out that the difficulty of managing today's computing systems goes well beyond the administration of individual software environments. The need to integrate several heterogeneous environments into corporate-wide computing systems, and to extend that beyond company boundaries into the Internet, introduces new levels of complexity. Computing systems' complexity appears to be approaching the limits of human capability, yet the march toward increased interconnectivity and integration rushes ahead unabated.

This march could turn the dream of pervasive computing—trillions of computing devices connected to the Internet—into a nightmare. Programming language innovations have extended the size and complexity of systems that architects can design, but relying solely on further innovations in programming methods will not get us through the present complexity crisis.

As systems become more interconnected and diverse, architects are less able to anticipate and design interactions among components, leaving such issues to be dealt with at runtime. Soon systems will become too massive and complex for even the most skilled system integrators to install, configure, optimize, maintain, and merge. And there will be no way to make timely, decisive responses to the rapid stream of changing and conflicting demands.

AUTONOMIC OPTION

The only option remaining is *autonomic computing*—computing systems that can manage themselves given high-level objectives from administrators. When IBM's senior vice president of research, Paul Horn, introduced this idea to the National Academy of Engineers at Harvard University in a March 2001 keynote address, he deliberately chose a term with a biological connotation. The autonomic nervous system governs our heart rate and body temperature, thus freeing our conscious brain from the burden of dealing with these and many other low-level, yet vital, functions.

The term autonomic computing is emblematic of a vast and somewhat tangled hierarchy of natural self-governing systems, many of which consist of myriad interacting, self-governing components that in turn comprise large numbers of interacting, autonomous, self-governing components at the next level down. The enormous range in scale, starting with molecular machines within cells and extending to human markets, societies, and the entire world socioeconomy, mirrors that of computing systems, which run from individual devices to the entire Internet. Thus, we believe it will be profitable to seek inspiration in the self-governance of social and economic systems as well as purely biological ones.

Clearly then, autonomic computing is a grand

0018-9162/03/$17.00 © 2003 IEEE

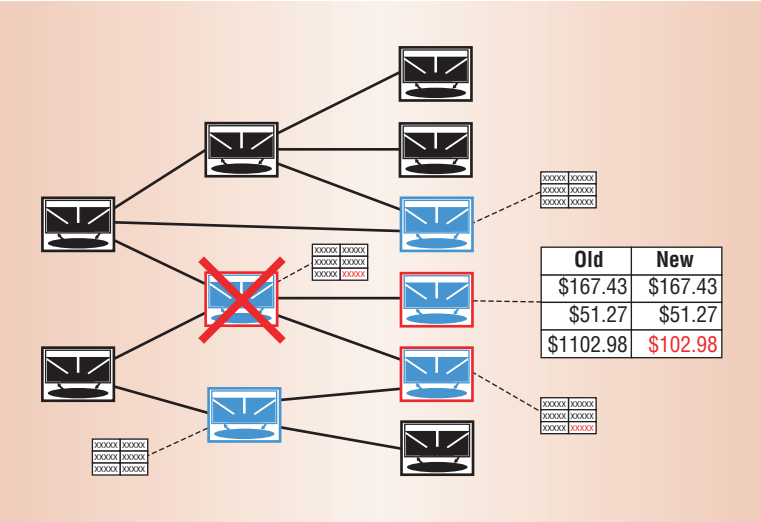


Figure 1. Problem diagnosis in an autonomic system upgrade. The upgrade introduces five software modules (blue), each an autonomic element. Minutes after installation, regression testers find faulty output in three of the new modules (red outlines), and the system immediately reverts to its old version. A problem determiner, an autonomic element, obtains information about interelement dependencies (lines between elements) from a dependency analyzer, another autonomic element that probes the system periodically (not shown). Taking into account its knowledge of interelement dependencies, the problem determiner analyzes log files and infers which of the three potentially bad modules is the culprit (red X). It generates a problem ticket containing diagnostic information and sends it to a software developer, who debugs the module and makes it available for future upgrades.

challenge that reaches far beyond a single organization. Its realization will take a concerted, long-term, worldwide effort by researchers in a diversity of fields. A necessary first step is to examine this vision: what autonomic computing systems might look like, how they might function, and what obstacles researchers will face in designing them and understanding their behavior.

SELF-MANAGEMENT

The essence of autonomic computing systems is self-management, the intent of which is to free system administrators from the details of system operation and maintenance and to provide users with a machine that runs at peak performance 24/7.

Like their biological namesakes, autonomic systems will maintain and adjust their operation in the face of changing components, workloads, demands, and external conditions and in the face of hardware or software failures, both innocent and malicious. The autonomic system might continually monitor its own use, and check for component upgrades, for example. If it deems the advertised features of the upgrades worthwhile, the system will install them, reconfigure itself as necessary, and run a regression test to make sure all is well. When it detects errors, the system will revert to the older version while its automatic problem-determination algorithms try to isolate the source of the error. Figure 1 illustrates how this process might work for an autonomic accounting system upgrade.

IBM frequently cites four aspects of self-management, which Table 1 summarizes. Early autonomic systems may treat these aspects as distinct, with different product teams creating solutions that address each one separately. Ultimately, these aspects will be emergent properties of a general architecture, and distinctions will blur into a more general notion of self-maintenance.

The journey toward fully autonomic computing will take many years, but there are several important and valuable milestones along the path. At first, automated functions will merely collect and aggregate information to support decisions by human administrators. Later, they will serve as advisors, suggesting possible courses of action for humans to consider. As automation technologies improve, and our faith in them grows, we will entrust autonomic systems with making—and acting on—lower-level decisions. Over time, humans will need to make relatively less frequent predominantly higher-level decisions, which the system will carry out automatically via more numerous, lower-level decisions and actions.

Ultimately, system administrators and end users will take the benefits of autonomic computing for granted. Self-managing systems and devices will seem completely natural and unremarkable, as will automated software and middleware upgrades. The detailed migration patterns of applications or data will be as uninteresting to us as the details of routing a phone call through the telephone network.

Self-configuration

Installing, configuring, and integrating large, complex systems is challenging, time-consuming, and error-prone even for experts. Most large Web sites and corporate data centers are haphazard accretions of servers, routers, databases, and other technologies on different platforms from different vendors. It can take teams of expert programmers months to merge two systems or to install a major e-commerce application such as SAP.

Autonomic systems will configure themselves automatically in accordance with high-level policies—representing business-level objectives, for

Table 1. Four aspects of self-management as they are now and would be with autonomic computing.

Concept	Current computing	Autonomic computing
Self-configuration	Corporate data centers have multiple vendors and platforms. Installing, configuring, and integrating systems is time consuming and error prone.	Automated configuration of components and systems follows high-level policies. Rest of system adjusts automatically and seamlessly.
Self-optimization	Systems have hundreds of manually set, nonlinear tuning parameters, and their number increases with each release.	Components and systems continually seek opportunities to improve their own performance and efficiency.
Self-healing	Problem determination in large, complex systems can take a team of programmers weeks.	System automatically detects, diagnoses, and repairs localized software and hardware problems.
Self-protection	Detection of and recovery from attacks and cascading failures is manual.	System automatically defends against malicious attacks or cascading failures. It uses early warning to anticipate and prevent systemwide failures.

example—that specify what is desired, not how it is to be accomplished. When a component is introduced, it will incorporate itself seamlessly, and the rest of the system will adapt to its presence—much like a new cell in the body or a new person in a population. For example, when a new component is introduced into an autonomic accounting system, as in Figure 1, it will automatically learn about and take into account the composition and configuration of the system. It will register itself and its capabilities so that other components can either use it or modify their own behavior appropriately.

Self-optimization

Complex middleware, such as WebSphere, or database systems, such as Oracle or DB2, may have hundreds of tunable parameters that must be set correctly for the system to perform optimally, yet few people know how to tune them. Such systems are often integrated with other, equally complex systems. Consequently, performance-tuning one large subsystem can have unanticipated effects on the entire system.

Autonomic systems will continually seek ways to improve their operation, identifying and seizing opportunities to make themselves more efficient in performance or cost. Just as muscles become stronger through exercise, and the brain modifies its circuitry during learning, autonomic systems will monitor, experiment with, and tune their own parameters and will learn to make appropriate choices about keeping functions or outsourcing them. They will proactively seek to upgrade their function by finding, verifying, and applying the latest updates.

Self-healing

IBM and other IT vendors have large departments devoted to identifying, tracing, and determining the root cause of failures in complex computing systems. Serious customer problems can take teams of programmers several weeks to diagnose and fix, and sometimes the problem disappears mysteriously without any satisfactory diagnosis.

Autonomic computing systems will detect, diagnose, and repair localized problems resulting from bugs or failures in software and hardware, perhaps through a regression tester, as in Figure 1. Using knowledge about the system configuration, a problem-diagnosis component (based on a Bayesian network, for example) would analyze information from log files, possibly supplemented with data from additional monitors that it has requested. The system would then match the diagnosis against known software patches (or alert a human programmer if there are none), install the appropriate patch, and retest.

Self-protection

Despite the existence of firewalls and intrusion-detection tools, humans must at present decide how to protect systems from malicious attacks and inadvertent cascading failures.

Autonomic systems will be self-protecting in two senses. They will defend the system as a whole against large-scale, correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures. They also will anticipate problems based on early reports from sensors and take steps to avoid or mitigate them.

ARCHITECTURAL CONSIDERATIONS

Autonomic systems will be interactive collections of *autonomic elements*—individual system constituents that contain resources and deliver services to humans and other autonomic elements. Autonomic elements will manage their internal behavior and their relationships with other autonomic elements in accordance with policies that humans or other elements have established. *System* self-management will arise at least as much from the myriad

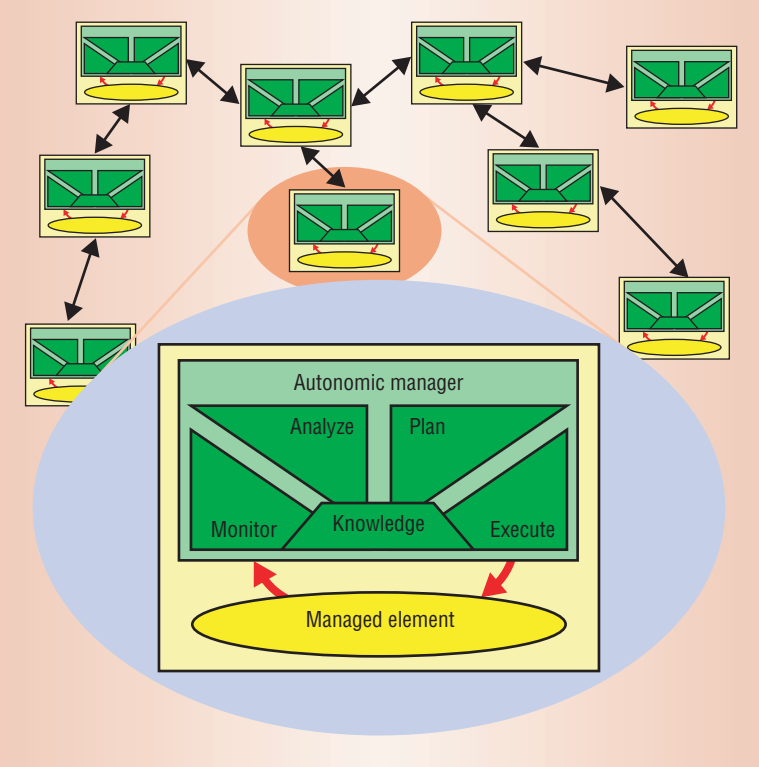


Figure 2. Structure of an autonomic element. Elements interact with other elements and with human programmers via their autonomic managers.

interactions among autonomic elements as it will from the internal self-management of the individual autonomic elements—just as the social intelligence of an ant colony arises largely from the interactions among individual ants. A distributed, service-oriented infrastructure will support autonomic elements and their interactions.

As Figure 2 shows, an autonomic element will typically consist of one or more managed elements coupled with a single autonomic manager that controls and represents them. The managed element will essentially be equivalent to what is found in ordinary nonautonomic systems, although it can be adapted to enable the autonomic manager to monitor and control it. The managed element could be a hardware resource, such as storage, a CPU, or a printer, or a software resource, such as a database, a directory service, or a large legacy system.

At the highest level, the managed element could be an e-utility, an application service, or even an individual business. The autonomic manager distinguishes the autonomic element from its nonautonomic counterpart. By monitoring the managed element and its external environment, and constructing and executing plans based on an analysis of this information, the autonomic manager will relieve humans of the responsibility of directly managing the managed element.

Fully autonomic computing is likely to evolve as designers gradually add increasingly sophisticated autonomic managers to existing managed elements. Ultimately, the distinction between the autonomic manager and the managed element may become merely conceptual rather than architectural, or it may melt away—leaving fully integrated, autonomic elements with well-defined behaviors and interfaces, but also with few constraints on their internal structure.

Each autonomic element will be responsible for managing its own internal state and behavior and for managing its interactions with an environment that consists largely of signals and messages from other elements and the external world. An element's internal behavior and its relationships with other elements will be driven by goals that its designer has embedded in it, by other elements that have authority over it, or by subcontracts to peer elements with its tacit or explicit consent. The element may require assistance from other elements to achieve its goals. If so, it will be responsible for obtaining necessary resources from other elements and for dealing with exception cases, such as the failure of a required resource.

Autonomic elements will function at many levels, from individual computing components such as disk drives to small-scale computing systems such as workstations or servers to entire automated enterprises in the largest autonomic system of all—the global economy.

At the lower levels, an autonomic element's range of internal behaviors and relationships with other elements, and the set of elements with which it can interact, may be relatively limited and hard-coded. Particularly at the level of individual components, well-established techniques—many of which fall under the rubric of fault tolerance—have led to the development of elements that rarely fail, which is one important aspect of being autonomic. Decades of developing fault-tolerance techniques have produced such engineering feats as the IBM zSeries servers, which have a mean time to failure of several decades.

At the higher levels, fixed behaviors, connections, and relationships will give way to increased dynamism and flexibility. All these aspects of autonomic elements will be expressed in more high-level, goal-oriented terms, leaving the elements themselves with the responsibility for resolving the details on the fly.

Hard-coded behaviors will give way to behaviors expressed as high-level objectives, such as "maximize this utility function," or "find a reputable message translation service." Hardwired connections among elements will give way to increasingly less direct specifications of an element's partners—from specification by physical address to specification by name and finally to specification by function, with the partner's identity being resolved only when it is needed.

Hard-wired relationships will evolve into flexible relationships that are established via negotiation. Elements will automatically handle new modes of failure, such as contract violation by a supplier, without human intervention.

While service-oriented architectural concepts like Web and grid services[2,3] will play a fundamental role, a sufficient foundation for autonomic computing requires more. First, as service providers, autonomic elements will not unquestioningly honor requests for service, as would typical Web services or objects in an object-oriented environment. They will provide a service only if providing it is consistent with their goals. Second, as consumers, autonomic elements will autonomously and proactively issue requests to other elements to carry out their objectives.

Finally, autonomic elements will have complex life cycles, continually carrying on multiple threads of activity, and continually sensing and responding to the environment in which they are situated. Autonomy, proactivity, and goal-directed interactivity with their environment are distinguishing characteristics of software agents. Viewing autonomic elements as agents and autonomic systems as multiagent systems makes it clear that agent-oriented architectural concepts will be critically important.[4]

ENGINEERING CHALLENGES

Virtually every aspect of autonomic computing offers significant engineering challenges. The life cycle of an individual autonomic element or of a relationship among autonomic elements reveals several challenges. Others arise in the context of the system as a whole, and still more become apparent at the interface between humans and autonomic systems.

Life cycle of an autonomic element

An autonomic element's life cycle begins with its design and implementation; continues with test and verification; proceeds to installation, configuration, optimization, upgrading, monitoring, problem determination, and recovery; and culminates in uninstallation or replacement. Each of these stages has special issues and challenges.

Design, test, and verification. Programming an autonomic element will mean extending Web services or grid services with programming tools and techniques that aid in managing relationships with other autonomic elements. Because autonomic elements both consume and provide services, representing needs and preferences will be just as important as representing capabilities. Programmers will need tools that help them acquire and represent policies—high-level specifications of goals and constraints, typically represented as rules or utility functions—and map them onto lower-level actions. They will also need tools to build elements that can establish, monitor, and enforce agreements.

Autonomic elements will provide a service only if it is consistent with their goals.

Testing autonomic elements and verifying that they behave correctly will be particularly challenging in large-scale systems because it will be harder to anticipate their environment, especially when it extends across multiple administrative domains or enterprises. Testing networked applications that require coordinated interactions among several autonomic elements will be even more difficult.

It will be virtually impossible to build test systems that capture the size and complexity of realistic systems and workloads. It might be possible to test newly deployed autonomic elements in situ by having them perform alongside more established and trusted elements with similar functionality.

The element's potential customers may also want to test and verify its behavior, both before establishing a service agreement and while the service is provided. One approach is for the autonomic element to attach a testing method to its service description.

Installation and configuration. Installing and configuring autonomic elements will most likely entail a bootstrapping process that begins when the element registers itself in a directory service by publishing its capabilities and contact information. The element might also use the directory service to discover suppliers or brokers that may provide information or services it needs to complete its initial configuration. It can also use the service to seek out potential customers or brokers to which it can delegate the task of finding customers.

Monitoring and problem determination. Monitoring will be an essential feature of autonomic elements. Elements will continually monitor themselves to ensure that they are meeting their own objectives, and they will log this information to serve as the

The vision of autonomic systems as a complex supply web makes problem determination both easier and harder than it is now.

basis for adaptation, self-optimization, and reconfiguration. They will also continually monitor their suppliers to ensure that they are receiving the agreed-on level of service and their customers to ensure that they are not exceeding the agreed-on level of demand. Special sentinel elements may monitor other elements and issue alerts to interested parties when they fail.

When coupled with event correlation and other forms of analysis, monitoring will be important in supporting problem determination and recovery when a fault is found or suspected. Applying monitoring, audit, and verification tests at all the needed points without burdening systems with excessive bandwidth or processing demands will be a challenge. Technologies to allow statistical or sample-based testing in a dynamic environment may prove helpful.

The vision of autonomic systems as a complex supply web makes problem determination both easier and harder than it is now. An autonomic element that detects poor performance or failure in a supplier may not attempt a diagnosis; it may simply work around the problem by finding a new supplier.

In other situations, however, it will be necessary to determine why one or more elements are failing, preferably without shutting down and restarting the entire system. This requires theoretically grounded tools for tracing, simulation, and problem determination in complex dynamic environments. Particularly when autonomic elements—or applications based on interactions among multiple elements—have a large amount of state, recovering gracefully and quickly from failure or restarting applications after software has been upgraded or after a function has been relocated to new machines will be challenging. David Patterson and colleagues at the University of California, Berkeley, and Stanford University have made a promising start in this direction.[5]

Upgrading. Autonomic elements will need to upgrade themselves from time to time. They might subscribe to a service that alerts them to the availability of relevant upgrades and decide for themselves when to apply the upgrade, possibly with guidance from another element or a human. Alternatively, the system could create afresh entirely new elements as part of a system upgrade, eliminating outmoded elements only after the new ones establish that they are working properly.

Managing the life cycle. Autonomic elements will typically be engaged in many activities simultaneously: participating in one or more negotiations at various phases of completion, proactively seeking inputs from other elements, and so on. They will need to schedule and prioritize their myriad activities, and they will need to represent their life cycle so that they can both reason about it and communicate it to other elements.

Relationships among autonomic elements

In its most dynamic and elaborate form, the service relationship among autonomic elements will also have a life cycle. Each stage of this life cycle engenders its own set of engineering challenges and standardization requirements.

Specification. An autonomic element must have associated with it a set of output services it can perform and a set of input services that it requires, expressed in a standard format so that other autonomic elements can understand it. Typically, the element will register with a directory service such as Universal Description, Discovery, and Integration[6] or an Open Grid Services Architecture (OGSA) registry,[3] providing a description of its capabilities and details about addresses and the protocols other elements or people can use to communicate with it.

Establishing standard service ontologies and a standard service description syntax and semantics that are sufficiently expressive for machines to interpret and reason about is an area of active research. The US Defense Advanced Research Projects Agency's semantic Web effort[7] is representative.

Location. An autonomic element must be able to locate input services that it needs; in turn, other elements that require its output services must be able to locate that element.

To locate other elements dynamically, the element can look them up by name or function in a directory service, possibly using a search process that involves sophisticated reasoning about service ontologies. The element can then contact one or more potential service providers directly and converse with them to determine if it can provide exactly the service they require.

In many cases, autonomic elements will also need to judge the likely reliability or trustworthiness of potential partners—an area of active research with many unsolved fundamental problems.

Negotiation. Once an element finds potential providers of an input service, it must negotiate with them to obtain that service.

We construe negotiation broadly as any process by which an agreement is reached. In *demand-for-service* negotiation, the element providing a service is subservient to the one requesting it, and the

provider must furnish the service unless it does not have sufficient resources to do so. Another simple form of negotiation is *first-come, first-served*, in which the provider satisfies all requests until it runs into resource limitations. In *posted-price* negotiation, the provider sets a price in real or artificial currency for its service, and the requester must take it or leave it.

More complex forms of negotiation include bilateral or multilateral negotiations over multiple attributes, such as price, service level, and priority, involving multiple rounds of proposals and counterproposals. A third-party arbiter can run an auction or otherwise assist these more complex negotiations, especially when they are multilateral.

Negotiation will be a rich source of engineering and scientific challenges for autonomic computing. Elements need flexible ways to express multiattribute needs and capabilities, and they need mechanisms for deriving these expressions from human input or from computation. They also need effective negotiation strategies and protocols that establish the rules of negotiation and govern the flow of messages among the negotiators. There must be languages for expressing service agreements—the culmination of successful negotiation—in their transient and final forms.

Efforts to standardize the representation of agreements are under way, but mechanisms for negotiating, enforcing, and reasoning about agreements are lacking, as are methods for translating them into action plans.

Provision. Once two elements reach an agreement, they must provision their internal resources. Provision may be as simple as noting in an access list that a particular element can request service in the future, or it may entail establishing additional relationships with other elements, which become subcontractors in providing some part of the agreed-on service or task.

Operation. Once both sides are properly provisioned, they operate under the negotiated agreement. The service provider's autonomic manager oversees the operation of its managed element, monitoring it to ensure that the agreement is being honored; the service requester might similarly monitor the level of service.

If the agreement is violated, one or both elements would seek an appropriate remedy. The remedy may be to assess a penalty, renegotiate the agreement, take technical measures to minimize any harm from the failure, or even terminate the agreement.

Termination. When the agreement has run its course, the parties agree to terminate it, freeing their internal resources for other uses and terminating agreements for input services that are no longer needed. The parties may record pertinent information about the service relationship locally, or store it in a database a reputation element maintains.

Systemwide issues

Other important engineering issues that arise at the system level include security, privacy, and trust, and the emergence of new types of services to serve the needs of other autonomic elements.

System-level engineering issues include security, privacy, and trust, and new types of services to serve the needs of other autonomic elements.

Autonomic computing systems will be subject to all the security, privacy, and trust issues that traditional computing systems must now address. Autonomic elements and systems will need to both establish and abide by security policies, just as human administrators do today, and they will need to do so in an understandable and fail-safe manner.

Systems that span multiple administrative domains—especially those that cross company boundaries—will face many of the challenges that now confront electronic commerce. These include authentication, authorization, encryption, signing, secure auditing and monitoring, nonrepudiation, data aggregation and identity masking, and compliance with complex legal requirements that vary from state to state or country to country.

The autonomic systems infrastructure must let autonomic elements identify themselves, verify the identities of other entities with which they communicate, verify that a message has not been altered in transit, and ensure that unauthorized parties do not read messages and other data. To satisfy privacy policies and laws, elements must also appropriately protect private and personal information that comes into their possession. Measures that keep data segregated according to its origin or its purpose must be extended into the realm of autonomic elements to satisfy policy and legal requirements.

Autonomic systems must be robust against new and insidious forms of attack that use self-management based on high-level policies to their own advantage. By altering or otherwise manipulating high-level policies, an attacker could gain much greater leverage than is possible in nonautonomic systems. Preventing such problems may require a new subfield of computer security that seeks to thwart fraud and the fraudulent persuasion of autonomic elements.

On a larger scale, autonomic elements will be

To satisfy privacy policies and laws, elements must appropriately protect information that comes into their possession.

agents, and autonomic systems will in effect be multiagent systems built on a Web services or OGSA infrastructure. Autonomic systems will be inhabited by middle agents[8] that serve as intermediaries of various types, including directory services, matchmakers, brokers, auctioneers, data aggregators, dependency managers—for detecting, recording, and publicizing information about functional dependencies among autonomic elements—event correlators, security analysts, time-stampers, sentinels, and other types of monitors that assess the health of other elements or of the system as a whole.

Traditionally, many of these services have been part of the system infrastructure; in a multiagent, autonomic world, moving them out of the infrastructure and representing them as autonomic elements themselves will be more natural and flexible.

Goal specification

While autonomic systems will assume much of the burden of system operation and integration, it will still be up to humans to provide those systems with policies—the goals and constraints that govern their actions. The enormous leverage of autonomic systems will greatly reduce human errors, but it will also greatly magnify the consequences of any error humans do make in specifying goals.

The indirect effect of policies on system configuration and behavior exacerbates the problem because tracing and correcting policy errors will be very difficult. It is thus critical to ensure that the specified goals represent what is really desired. Two engineering challenges stem from this mandate: Ensure that goals are specified correctly in the first place, and ensure that systems behave reasonably even when they are not.

In many cases, the set of goals to be specified will be complex, multidimensional, and conflicting. Even a goal as superficially simple as "maximize utility" will require a human to express a complicated multiattribute utility function. A key to reducing error will be to simplify and clarify the means by which humans express their goals to computers. Psychologists and computer scientists will need to work together to strike the right balance between overwhelming humans with too many questions or too much information and underempowering them with too few options or too little information.

The second challenge—ensuring reasonable system behavior in the face of erroneous input—is another facet of robustness: Autonomic systems will need to protect themselves from input goals that are inconsistent, implausible, dangerous, or unrealizable with the resources at hand. Autonomic systems will subject such inputs to extra validation, and when self-protective measures fail, they will rely on deep-seated notions of what constitutes acceptable behavior to detect and correct problems. In some cases, such as resource overload, they will inform human operators about the nature of the problem and offer alternative solutions.

SCIENTIFIC CHALLENGES

The success of autonomic computing will hinge on the extent to which theorists can identify universal principles that span the multiple levels at which autonomic systems can exist—from systems to enterprises to economies.

Behavioral abstractions and models

Defining appropriate abstractions and models for understanding, controlling, and designing emergent behavior in autonomic systems is a challenge at the heart of autonomic computing. We need fundamental mathematical work aimed at understanding how the properties of self-configuration, self-optimization, self-maintenance, and robustness arise from or depend on the behaviors, goals, and adaptivity of individual autonomic elements; the pattern and type of interactions among them; and the external influences or demands on the system.

Understanding the mapping from local behavior to global behavior is a necessary but insufficient condition for controlling and designing autonomic systems. We must also discover how to exploit the inverse relationship: How can we derive a set of behavioral and interaction rules that, if embedded in individual autonomic elements, will induce a desired global behavior? The nonlinearity of emergent behavior makes such an inversion highly nontrivial.

One plausible approach couples advanced search and optimization techniques with parameterized models of the local-to-global relationship and the likely set of environmental influences to which the system will be subjected. Melanie Mitchell and colleagues[9] at the Santa Fe Institute have pioneered this approach, using genetic algorithms to evolve the local transformation rules of simple cellular automata to achieve desired global behaviors. At NASA, David Wolpert and colleagues[10] have studied algorithms that, given a high-level global objective, derive individual goals for individual agents. When each agent selfishly follows its goals, the desired global behavior results.

These methods are just a start. We have yet to understand fundamental limits on what classes of

global behavior can be achieved, nor do we have practical methods for designing emergent system behavior. Moreover, although these methods establish the rules of a system at design time, autonomic systems must deal with shifting conditions that can be known only at runtime. Control theoretic approaches may prove useful in this capacity; some autonomic managers may use control systems to govern the behavior of their associated managed elements.

The greatest value may be in extending distributed or hierarchical control theories, which consider interactions among independently or hierarchically controlled elements, rather than focusing on an individual controlled element. Newer paradigms for control may be needed when there is no clear separation of scope or time scale.

Robustness theory

A related challenge is to develop a theory of robustness for autonomic systems, including definitions and analyses of robustness, diversity, redundancy, and optimality and their relationship to one another. The Santa Fe Institute recently began a multidisciplinary study on this topic (http://discuss.santafe.edu/robustness).

Learning and optimization theory

Machine learning by a single agent in relatively static environments is well studied, and it is well supported by strong theoretical results. However, in more sophisticated autonomic systems, individual elements will be agents that continually adapt to their environment—an environment that consists largely of other agents. Thus, even with stable external conditions, agents are adapting to one another, which violates the traditional assumptions on which single-agent learning theories are based.

There are no guarantees of convergence. In fact, interesting forms of instability have been observed in such cases.[11] Learning in multiagent systems is a challenging but relatively unexplored problem, with virtually no major theorems and only a handful of empirical results.

Just as learning becomes a more challenging problem in multiagent systems, so does optimization. The root cause is the same—whether it is because they are learning or because they are optimizing, agents are changing their behavior, making it necessary for other agents to change their behavior, potentially leading to instabilities. Optimization in such an environment must deal with dynamics created by a collective mode of oscillation rather than a drifting environmental signal. Optimization techniques that assume a stationary environment have been observed to fail pathologically in multiagent systems,[12] therefore they must either be revamped or replaced with new methods.

Optimization techniques that assume a stationary environment must either be revamped or replaced with new methods.

Negotiation theory

A solid theoretical foundation for negotiation must take into account two perspectives. From the perspective of individual elements, we must develop and analyze algorithms and negotiation protocols and determine what bidding or negotiation algorithms are most effective.

From the perspective of the system as a whole, we must establish how overall system behavior depends on the mixture of negotiation algorithms that various autonomic elements use and establish the conditions under which multilateral—as opposed to bilateral—negotiations among elements are necessary or desirable.

Automated statistical modeling

Statistical models of large networked systems will let autonomic elements or systems detect or predict overall performance problems from a stream of sensor data from individual devices. At long time scales—during which the configuration of the system changes—we seek methods that automate the aggregation of statistical variables to reduce the dimensionality of the problem to a size that is amenable to adaptive learning and optimization techniques that operate on shorter time scales.

Is it possible to meet the grand challenge of autonomic computing without magic and without fully solving the AI problem? We believe it is, but it will take time and patience. Long before we solve many of the more challenging problems, less automated realizations of autonomic systems will be extremely valuable, and their value will increase substantially as autonomic computing technology improves and earns greater trust and acceptance.

A vision this large requires that we pool expertise in many areas of computer science as well as in disciplines that lie far beyond computing's traditional boundaries. We must look to scientists studying nonlinear dynamics and complexity for new theories of emergent phenomena and robustness. We must look to economists and e-commerce researchers for ideas and technologies about negotiation and supply webs. We must look to psychologists and human factors researchers for new goal-definition and visualization paradigms and

for ways to help humans build trust in autonomic systems. We must look to the legal profession, since many of the same issues that arise in the context of e-commerce will be important in autonomic systems that span organizational or national boundaries.

Bridging the language and cultural divides among the many disciplines needed for this endeavor and harnessing the diversity to yield successful and perhaps universal approaches to autonomic computing will perhaps be the greatest challenge. It will be interesting to see what new cross-disciplines develop as we begin to work together to solve these fundamental problems. ■

Acknowledgments

We are indebted to the many people who influenced this article with their ideas and thoughtful criticisms. Special thanks go to David Chambliss for contributing valuable thoughts on human-computer interface issues. We also thank Bill Arnold, David Bantz, Rob Barrett, Peter Capek, Alan Ganek, German Goldszmidt, James Hanson, Joseph Hellerstein, James Kozloski, Herb Lee, Charles Peck, Ed Snible, and Ian Whalley for their helpful comments, and the members of an IBM Academy of Technology team for their extensive written and verbal contributions: Lisa Spainhower and Kazuo Iwano (co-leaders), William H. Tetzlaff (Technology Council contact), Robert Abrams, Sam Adams, Steve Burbeck, Bill Chung, Denise Y. Dyko, Stuart Feldman, Lorraine Herger, Mark Johnson, James Kaufman, David Kra, Ed Lassettre, Andreas Maier, Timothy Marchini, Norm Pass, Colin Powell, Stephen A. Smithers, Daniel Sturman, Mark N. Wegman, Steve R. White, and Daniel Yellin.

References

1. IBM, "Autonomic Computing: IBM's Perspective on the State of Information Technology"; http://www-1.ibm.com/industries/government/doc/content/resource/thought/278606109.html.
2. H. Kreger, "Web Services Conceptual Architecture," v. 1.0. 2001; http://www-4.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.
3. I. Foster et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Feb. 2002; http://www.globus.org/research/papers/ogsa.pdf.
4. N.R. Jennings, "On Agent-Based Software Engineering," *Artificial Intelligence*, vol. 177, no. 2, 2000, pp. 277-296.
5. D. Patterson et al., *Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies*, tech. report CSD-02-1175, Computer Science Dept., Univ. of Calif., Berkeley, Calif., Mar. 2002.
6. Ariba, IBM, and Microsoft, "UDDI Technical White Paper," 2000; http://www.uddi.org/whitepapers.html.
7. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, May 2001, pp. 28-37.
8. H. Wong and K. Sycara, "A Taxonomy of Middle Agents for the Internet," *Proc. 4th Int'l Conf. Multiagent Systems*, IEEE CS Press, 2000, pp. 465-466.
9. R. Das et al., "Evolving Globally Synchronized Cellular Automata," *Proc. 6th Int'l Conf. Genetic Algorithms*, L. Eshelman, ed., Morgan Kaufmann, 1995, pp. 336-343.
10. D. Wolpert, K. Wheeler, and K. Tumer, *Collective Intelligence for Control of Distributed Dynamical Systems*, tech. report NASA-ARC-IC-99-44, NASA, Ames, Iowa, 1999.
11. J.O. Kephart and G.J. Tesauro, "Pseudo-Convergent Q-Learning by Competitive Pricebots," *Proc. 17th Int'l Conf. Machine Learning*, Morgan Kaufmann, 2000, pp. 463-470.
12. J.O. Kephart et al., "Pricing Information Bundles in a Dynamic Environment," *Proc. 3rd ACM Conf. Electronic Commerce*, 2001, ACM Press, pp. 180-190.

Jeffrey O. Kephart *manages the Agents and Emergent Phenomena group at the IBM Thomas J. Watson Research Center. His research focuses on the application of analogies from biology and economics to massively distributed computing systems, particularly in the domains of autonomic computing, e-commerce, and antivirus technology. Kephart received a BS from Princeton University and a PhD from Stanford University, both in electrical engineering. Contact him at kephart@us.ibm.com.*

David M. Chess *is a research staff member at the IBM Thomas J. Watson Research Center, working in autonomic computing and computer security. He received a BA in philosophy from Princeton University and an MS in computer science from Pace University. Contact him at chess@us.ibm.com.*