RC23692 (W0508-070) August 9, 2005
Computer Science

IBM Research Report

Research Challenges of Autonomic Computing

Jeffrey O. Kephart
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Research Challenges of Autonomic Computing

Jeffrey O. Kephart
IBM Thomas J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532, USA
kephart@us.ibm.com

ABSTRACT

Autonomic computing is a grand-challenge vision of the future in which computing systems will manage themselves in accordance with high-level objectives specified by humans. The IT industry recognizes that meeting this challenge is imperative; otherwise, IT systems will soon become virtually impossible to administer. But meeting this challenge is also extremely difficult, and will require a worldwide collaboration among the best minds of academia and industry. In the hope of motivating researchers in relevant areas to apply their expertise to this vitally important problem, I outline some of the main scientific and engineering challenges that collectively make up the grand challenge of autonomic computing, and provide pointers to initial efforts to address these challenges.

Categories and Subject Descriptors

A.1 [**General Literature**]: Introduction and Survey; H.0 [**Information Systems**]: General; H.4 [**Information Systems Applications**]: General; I.O [**Computing Methodologies**]: General; I.2.11 [**Distributed Artificial Intelligence**]: [intelligent agents, multiagent systems]; K.6 [**Management of Computing and Information Systems**]: General

General Terms

Management, Measurement, Performance, Reliability

Keywords

Autonomic computing, self-managing systems, research challenges

1. INTRODUCTION

Typical present-day IT[1] environments are complex, heterogeneous tangles of hardware, middleware and software from multiple vendors that are becoming increasingly difficult to integrate, install, configure, tune, and maintain. At the present rate of growth in complexity, even the most skilled IT professionals may find it impossible to administer IT environments within a few years. Most of the IT industry recognizes that the only viable solution to this looming crisis is to endow systems and the components that comprise them with the ability to manage themselves in accordance with high-level objectives specified by humans [38]. IBM introduced this vision of self-managing systems in 2001 when it launched the Autonomic Computing initative [24]. Hewlett-Packard's *Adaptive Enterprise* initiative [23] and Microsoft's *Dynamic Systems* initiative [44] are related industry efforts that recognize that self-managing components and systems are vital to the future of IT.

The purpose of this paper is to decompose the grand challenge of autonomic computing into several of its constituent scientific and engineering challenges. Even incremental steps towards meeting some of these individual challenges are likely to be beneficial to industry in the near-term, long before the vision of autonomic computing is realized fully. As evidenced by over 20 workshops and conferences devoted to the topic over the last two years, autonomic computing offers a rich application domain for researchers in many branches of computer science, including software architecture, systems, artificial intelligence, and human-computer interfaces. It is also likely to benefit from advances in applied mathematics, novel applications of economic mechanisms, and even ethnographic studies. It is hoped that this brief survey of autonomic computing research challenges will encourage researchers in relevant areas to join this movement, and lend their expertise to problems whose solution would have an enormous impact on the IT industry.

The paper is organized as follows. First, in section 2, I describe a framework that has proved to be useful in defining and describing IBM's autonomic computing research program. Then, in sections 3, 4, and 5, I use this framework to highlight a subset of challenges that define fruitful avenues for research—the pursuit of which could yield significant practical benefits in the intermediate to long term. I close with a summary and some final comments in section 6.

2. FRAMEWORK

Autonomic computing draws upon an enormous diversity of fields within and beyond the boundaries of traditional computer science. Therefore, any attempt to categorize and describe the whole research effort and its associated challenges is bound to be imperfect. The architecturally-flavored framework that I use in this paper has proven to be helpful in defining and growing IBM's own autonomic computing research program, which now comprises the diverse efforts of at least 100 researchers, and can thus be viewed as a microcosm of the broader worldwide effort on self-managing systems.

At the very coarsest level, I divide the research space into three basic parts: autonomic elements, autonomic systems, and human-computer interactions.

Autonomic elements are the basic building blocks of autonomic

[1] Information Technology

systems, which through their mutual interactions produce the overall self-managing behavior of autonomic computing systems. Essentially any type of computing resource can be viewed as an autonomic element: a storage device, a database or application server, a middleware component, a load balancer, a workload manager, a resource broker, etc. One can think of these autonomic elements as services within a service-oriented architecture. Alternatively, and in many cases more accurately, one can identify autonomic elements as software agents, and autonomic computing systems as multiagent systems. For more detail on autonomic elements, their behavior, and their interrelationships, see reference [38].

Within the autonomic element branch of the research framework, I distinguish three sub-branches:

- **Specific autonomic elements**. Research directed towards improving the self-managing capability of specific components such as databases, storage systems, servers, etc.
- **Generic autonomic element technologies**. Research on technologies that are generally applicable to autonomic elements, including planning, modeling, forecasting, optimization, etc.
- **Generic autonomic element architectures, tools, and prototypes**. Research on the internal structure of autonomic elements, tools to help create autonomic elements, and reference implementations of autonomic elements built with these tools.

Within the autonomic systems branch of the research framework, I distinguish three sub-branches:

- **Autonomic system technologies**. Research on generic technologies that entail interactions among multiple autonomic elements to achieve system-level goals, including problem determination and remediation, automated provisioning, workload management, automated installation and configuration, integrity management, etc.
- **Autonomic system architectures and prototypes**. Research on system-level architectures that effectively govern interactions among autonomic elements, and prototypes that assemble the many other pieces described in this framework to demonstrate systems with improved self-management relative to existing systems.
- **Autonomic system science**. Research on fundamental science of large-scale autonomic computing systems, addressing questions of learning, stability, control and emergent behavior in multiagent systems, and also addressing questions of how to quantify the degree of self-management in systems.

Finally, within the human interaction branch of the research framework, I distinguish two sub-branches:

- **Human studies**. Research on present and future interactions between human administrators and other users and self-managing systems, to determine what interfaces and other modes of interaction are most effective.
- **Policy**. Research on methods for eliciting high-level policies from people, representing and appropriately transforming those policies within autonomic systems, and managing behavior with respect to those policies.

In the remainder of this paper, I will structure a survey of research challenges in all of these areas around this framework, and I will provide pointers to initial efforts to address these challenges.

3. AUTONOMIC ELEMENT CHALLENGES

3.1 Specific autonomic elements

For many years, researchers have striven to make individual system components more self-managing, especially servers, database management systems, and storage systems. Certainly, extensive research programs in all three of these areas had been under way for decades at IBM and other research institutions when the autonomic computing initiative was first announced by Paul Horn in 2001. A steady stream of new self-managing capabilities has entered commercial products over the last several years, and trend we can expect to continue into the foreseeable future. The challenges within these individual domains are numerous but well-understood, and it would be presumptuous to attempt to catalog them here.

The main *new* research challenge introduced by the autonomic computing initiative is to achieve effective interoperation among autonomic elements. In order for this to happen, product developers must look beyond their natural product-centric tendencies and cultivate a more holistic, system-level point of view. In other words, specific autonomic elements must be designed with a greater awareness of the fact that they will be situated in autonomic systems and intercommunicating and interacting cooperatively with other autonomic elements. One requirement is that individual components must adopt standard interfaces defined by autonomic system architectures, as discussed further in Section 4.2. A second requirement is that individual components must generate and supply needed information to other components, and must be capable of requesting and using such information from other components. For example, a database component might need to answer what-if questions from application server middleware that is trying to add extra capacity to the application tier. Or an application server middleware component might be asked by a provisioning manager to estimate what would be the benefit of adding an additional server to the middle tier. The basis for this second set of capabilities will be discussed further in Sections 3.2 and 3.3.

3.2 Generic autonomic element technologies

Autonomic elements will draw upon a number of common technologies, including monitoring, event correlation, rule execution, modeling, optimization, forecasting, planning, feedback control, and machine learning. Here I outline special challenges that autonomic computing creates for some of these technologies.

Autonomic elements will need to share common methods for collecting and representing monitored data. The challenges here are mainly ones of standardization. One notable effort to define a consistent, common format for monitored events and log file formats is the Common Base Event format [25], which is being developed under the auspices of the OASIS Web Services Distributed Management Technical Committee. Legacy components can generate events and log files in their traditional formats, and a suitable Generic Log Adaptor can convert these events or log files into the CBE format, whereupon they can be stored in a common repository and/or sent to subscribers.

Rule engines and correlation engines (with their associated languages) are useful technologies for analyzing monitored data and log files to identify trends or situations that warrant deeper examination. One serious challenge in this space is to determine a good set of rules and/or correlation expressions that describe the conditions under which particular automated actions or analyses should be triggered, or humans should be alerted (presumably so that they can take action manually). One complication is that the conditions may require correlation across multiple components, so even assembling the requisite data in one place may be an issue. At the

very least, human experts are likely to need assistance in authoring a potentially large set of rules. Ideally, a good portion of the rule authoring should be based on some form of machine learning, and based on system-level goals coupled with historical observation. Promising initial efforts that automatically discover correlations between low-level system measurements and high-level Service Level Objectives include work by Hewlett-Packard Labs [53] and IBM Research [9].

The forementioned research on extracting correlations between low-level system measurements and high-level Service Level Objectives is based on statistical regression and/or modeling. Models, and methods for learning them automatically, are key to the functioning of autonomic elements in several respects. Autonomic elements need models that map potential actions into probable outcomes so that they can make decisions about the right action or action sequence. They need models of workload so that they can forecast future demand and plan accordingly. In some cases, they may need models of other system components with which they have relationships, or with which they are contemplating forming a relationship. And, as mentioned above, models are important because they establish useful relationships between the high-level terms in which humans wish to express goals and priorities for system behavior and the low-level control parameters and observables that autonomic elements can control and/or monitor. A major challenge for all types of modeling is to learn and re-adjust the models continually on the fly. Models must adjust quickly to observations with a minimum of data and training time, and they must adapt to conditions that are noisy and prone to fluctations that are extrinsic or intrinsic to the system. Queuing models, once used mainly for the essentially static role of capacity planning, are starting to be used for performance optimization and resource allocation, which require continual readjustment of queuing model parameters; the work of Menasce [3] and Chess et al. [17] exemplify this trend.

Three additional learning challenges are worth mentioning. First, many types of learning utilize random exploration to seek more profitable regions of parameter space. In simple classic machine learning problems, extensive trial and error is acceptable so long as the asymptotic performance is good. However, in autonomic systems, system components are unlikely to have such a luxury. They are expected to work well from the beginning and adapt themselves seamlessly to environmental changes. Too much trial and error may cause poor average performance. Thus the classic exploration vs. exploitation tradeoff is likely to be biased more towards exploitation. Finding principled ways to explore parameter space while still capitalizing on what has already been learned remains a challenge. Second, there may be several hundred tunable parameters in modern-day databases like Oracle and IBM DB2, or in web application server middleware. In order for any learning algorithm to train sufficiently quickly, this number must be reduced to no more than one or two dozen of the most relevant parameters. Identifying which are most relevant in a given situation is highly non-trivial. Third, traditional convergence guarantees that hold for machine learning by single agents in stationary environments [47] do not apply when multiple agents are learning simultaneously and their actions are coupled. The very act of learning by one agent changes the environment experienced by other agents, raising the possibility that the learning process may never converge [39]. The field of multiagent learning is still in its infancy, with a handful of techniques and even fewer theoretical analyses. The naïve approach of endowing autonomic elements with single-agent learning algorithms and hoping for convergence sometimes works, but there is a limited understanding of the conditions under which this approach is successful.

Optimization challenges overlap a good deal with those of learning. Autonomic elements inhabit a highly dynamic world. Regardless of whether they are using deterministic or stochastic approaches to optimization, they must cope with nonstationarity. In other words, due to either extrinsic effects such as changing workload or intrinsic effects such as adaptative behavior by other autonomic elements, an individual autonomic element's optimization landscape is bound to change over time. Many traditional optimization techniques assume implicitly that the objective function is static, and do not bother to resample it. This assumption has been observed to lead to pathological behavior in multiagent systems, even when the external environment is static, because adaptation by individual agents changes their behavior in a way that alters the objective functions of the other agents [37].

Over the last few years, the planning community has begun to recognize that, to be truly practical, planners must take several real-world issues into account, many of which are relevant to autonomic computing [32]. One real-world issue is that autonomic elements must conduct their planning in open environments that are not completely understood. This calls for planning techniques that gracefully handle incomplete domain specifications, i.e. the preconditions and/or postconditions for actions are not fully known. Moreover, since planning takes place in dynamic environments that may change during the course of executing a plan, it is vital to continually assess progress towards completion of a plan, and to replan whenever plans go awry. Finally, generating and manipulating plan metadata describing capabilities and conditions of use is very important, as this provides a way to catalog, reuse and possibly merge plans that have proven successful in the past. Several workshops are devoted to exploring these challenges, notably the *IJCAI-05 Workshop on Planning and Learning in* a Priori *Unknown or Dynamic Domains* and the *AAAI 2005 Workshop on Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing*.

3.3 Generic autonomic element architectures, tools, and prototypes

In addition to the challenges of developing individual autonomic element technologies, as outlined in Section 3.2, there is the challenge of bringing them all together coherently to create a single autonomic element that functions competently in an autonomic computing system. In their ultimate realization, autonomic elements will be software agents that concurrently and asynchronously execute multiple threads of activity that are directed by goals that are either intrinsic to the element or derived from service relationships with other elements [38]. A typical autonomic element might simultaneously be providing service to several other elements, consuming services from several others, and planning tasks associated with these service relationships. This challenge bears much in common with that of developing a competent software agent, which requires an effective marriage of technology with architecture.

A good autonomic element (or software agent) architecture should effectively support the need to manage and coordinate multiple threads of execution within the element. A single task such as honoring the terms of a service agreement with another element might entail monitoring the variables of interest, analyzing the extent to which they are being kept within agreed-upon bounds, and, if all is not satisfactory, invoking a planner to determine a sequence of actions that is likely to improve performance. Concurrently, the element may use the monitored data to improve its internal model of the relationship between performance and control parameters, or it may revise its forecast of future workload, or even the model upon which it bases such forecasts.

When one considers that there will be multiple tasks executing simultaneously, all of them requiring coordination across different monitoring, analysis, modeling, learning, optimization, and planning modules, it is clear that the challenge of developing an autonomic element architecture capable of managing such complexity is great. One fundamentally hard problem is to develop an architecture (and accompanying functionality) for detecting and resolving conflicts that are bound to occur when an autonomic element cannot meet all of its obligations. For example, consider an autonomic database element that needs to serve a given number of queries per minute, and which is required to run full backups every 24 hours. An optimizer may determine that the query process needs 25% more bandwidth and 10% more memory to perform satisfactorily, while a planner may determine that, in order to complete in time, the backup needs to start immediately and run at high priority until completion. These actions may or may not be in conflict with one another. Given that they are not expressed in the same terms, and given that the decisions behind them may not have been generated at exactly the same time, how is a possible conflict to be detected? And if a conflict exists, what mechanisms can be used to change one plan or the other (or both) so that the conflict is resolved?

Autonomic element architecture is important, but architecture alone will not suffice. Software developers need tools that provide a good set of autonomic element technologies, means for interconnecting them in accordance with the architecture, and easily integrable interfaces that permit the autonomic element to interact appropriately with other autonomic elements. (The details of what these interfaces should be are a matter of system-level architecture, to be discussed further in Section 4.2.) One early effort to provide some of this functionality to developers is the IBM Autonomic Computing toolkit [29], which is available for trial by software developers. It is encouraging that, despite the lack of many functions that one would like to have ultimately, the toolkit is already proving to be useful. An account of its use in developing self-configuring network services at Intel can be found in reference [43].

Finally, it is worth stating that building and refining prototypes, preferably ones grounded in real applications, is an essential part of the process of developing architectures and tools for autonomic elements. Simple autonomic elements built with the extended research version of the IBM Autonomic Computing Toolkit are described in reference [18], which also gives an account of how elements so constructed can interact to form a system that optimizes, configures, and heals itself. The process of using the tools to build architecturally-compliant elements provided several useful lessons that led to refinements in the tool and the architecture.

4. AUTONOMIC SYSTEM CHALLENGES

4.1 Autonomic system technologies

Technologies associated with system-level self-configuration, self-healing, self-optimization, and self-protection [38] tend to entail interactions among multiple autonomic elements.

First, consider self-configuration. Suppose there is an existing system to which a single new element (say a new server) is added. The challenge in this case is for the new element to integrate itself seamlessly into the system with minimal human intervention. In most schemes that have been proposed [18], this requires a form of bootstrapping in which the element interacts with a registry in two ways: it discovers other elements in the system that can help it complete its configuration, and it registers itself so that other system elements can find it and learn of its capabilities. Advances in OWL [19] and other efforts to develop Web Services semantics will help improve the ability of autonomic elements to express their needs, capabilities, and properties, and to reason about the needs, capabilities, and properties of other autonomic elements—enabling them to discover and use one another more effectively.

A more complex and challenging self-configuration scenario is the automated deployment of a large-scale application, which entails much more than merely running the self-configuration scenario of the previous paragraph several times. There are complex interdependencies among autonomic elements that must be taken into account in the detailed choreography that typifies large-scale solution deployment. The details of what servers will be chosen, how they will be arranged in multiple tiers, and where routers and firewalls will be placed will be derived automatically from a general specification of the application and its service level requirements, supplemented with knowledge of the existing infrastructure and appropriate metadata on the various system resources and their mutual interdependencies. To make this work in a realistic setting will require clever integration of capacity planning, capture and use of human expert knowledge about system configuration [21], and planning and scheduling technologies [35] that generate workflows describing the detailed installation and configuration choreography. A workflow so generated must be robust to changes in hardware and software license availability that may occur during the course of executing the workflow. This suggests that research on continual replanning and other new directions in practically-oriented planning will be important in this context [41]. Radically different approaches to self-configuration are worth exploring as well, such as the more decentralized paradigm of goal-directed self-assembly [18] or other ideas inspired by emergent computation and self-organizing systems [46].

Next, consider self-healing. The problem management lifecycle begins with a stage in which selected metrics of individual components are monitored and perhaps aggregated. In a subsequent stage, the stream of low-level events is processed into higher-level assessments of the health of individual components or the system as a whole. This is followed by problem localization and/or determination, and finally by problem remediation.

The first two stages, monitoring and analysis, have already been discussed somewhat in Section 3.2. Rule and correlation technologies are typically applied at the analysis stage because they can be applied efficiently to voluminous event streams. However, as discussed in section 3.2, the number of rules required can be large, and authoring is difficult. Any technologies that reduce the burden on humans to write low-level rules and correlation rules would be helpful (more detailed thoughts on human-computer interaction appear in Section 5.1.) In addition to developing machine learning techniques that automatically determine thresholds beyond which high-level events, actions, or alerts are generated, technologies that assist humans with rule authoring are needed, including new rule interfaces and mechanisms for capturing expert knowledge.

At the third stage, problem localization and/or determination, the challenges become especially interesting. Note that, while tracking a problem's exact origin down to the offending line of code is a wonderful ideal, even coarse localization to a bad router or unresponsive server can be quite valuable in autonomic systems. Even if an individual autonomic element cannot determine what ails it and heal itself, the system as a whole can still exhibit self-healing by replacing, rebooting, or otherwise working around the faulty component. Thus problem localization is a key ingredient of self-healing. In complex, highly interconnected systems, attributing the cause of a failure to a particular component can be tricky. For example, suppose you are connected to a web site and your session suddenly dies. The source of the problem could be practically

anywhere—on your machine (and even then it could be a hardware failure of a bug in any of a number of software modules), or it could be somewhere at the website (a hardware or software problem on any of a large number of servers or routers that support the web site), or it could even be somewhere in the network that connects you to that web site.

One approach to problem localization uses probe signals. Probe failures, such as a ping timeout or unresponsiveness to a test transaction, can be used to infer a set of possible candidate bad components, and that set can be narrowed down by using multiple overlapping probes. Since probes can create extra load on systems, they should be used as sparingly as possible. One interesting line of research [11] uses information theoretic techniques to identify optimally small and efficient sets or sequences of probes that can pinpoint problems; efforts are being made now to extend the work to scale to larger systems and handle multiple simultaneous failures. Problem localization techniques such as these rely on a knowledge of patterns of interconnection, or dependencies, among system components. This brings up yet another challenge that is beginning to be addressed [22]: how to extract these dependencies automatically as the system's configuration continually shifts?

The final stage, automated remediation of a problem once it has been localized, is perhaps the most difficult. One approach is to compile a large database of known symptoms and associated remedial actions. When a problem is localized and/or diagnosed, the symptom database is consulted, and if there is an adequate match to a known symptom then the associated action is taken. Here the main challenges are to populate the knowledge base and to develop methods for searching it effectively. Effective techniques for capturing human expert knowledge of problems would be very useful, as would automated techniques for harvesting logs to extract problem signatures, which Brodie and colleagues have begun to develop [10].

Another promising avenue of research on problem remediation is the recursive micro-rebooting concept of Fox et al. [14], which seeks to reengineer the rebooting bludgeon into a scalpel. In this scheme, software components at varying levels of granularity are designed to be crash-only, i.e. the only recourse if they are suspected of not functioning properly is to reboot them. Rather than rebooting an entire system or application, which could take 10 seconds to several minutes, a faulty Enterprise Java Bean (EJB) can be rebooted in mere tenths of a second. It remains to develop improved statistical techniques that distinguish clearly between working and failing states, as false positives lead to poor system performance due to excessive unnecessary reboots and false negatives leave the system in a nonfunctional state. Moreover, other remediation techniques must still be applied in cases where there are hard failures, configuration problems that cause persistent faults, or persistent state is corrupted.

Littman et al. [42] have taken a radically different approach that completely bypasses diagnosis and localization. They formulate autonomic network repair as a reinforcement learning problem in which numerous different fault types are injected into the network during training. The reinforcement learning algorithm explores several alternative test and repair actions. Over time, the algorithm learns which repair actions lead to the highest reward without learning any intermediate problem diagnosis representation. An important question is whether this technique can scale to larger, more complex systems in which the range of possible faults, elements, and remediation actions is significantly larger.

Next, consider self-optimization. Workload management and automated provisioning (or re-provisioning) are canonical examples of system-level self-optimization. Workload management seeks to tune control parameters (e.g. the proportion of memory or bandwidth allocated to a specific workload or job class) so as to provide the best possible service to a given workload or set of workloads, taking into account any tradeoffs that have been specified in policies. Automated provisioning works in conjunction with workload management to provide additional resources (e.g. new servers) when they are needed to meet performance objectives, and to remove those resources when they are needed elsewhere more urgently. In both cases, decisions about control parameter settings or resource allocation must be based on objectives (or policies) specified by humans. Performance models (as discussed in Section 3.2) play a key role in this context, as they establish a relationship between the control parameter settings or resource allocation and the expected service performance level in which policies tend to be expressed. Machine learning techniques are likely to play an increasingly prominently role in self-optimization, building on early results by Stone et al. [51, 52].

Perhaps the most difficult challenge in system-level system optimization is one of coordination. Database, storage, workload management, provisioning, and other self-optimizing elements all have their own independent means for expressing optimization criteria and for tuning themselves in accordance with those criteria, and these criteria are often mutually incompatible, making it impossible to coherently express and manage to system-level goals. There is an urgent need to establish a *lingua franca* for expressing optimization goals in self-managing systems, a point that will be discussed further in Section 5.2. Further compounding the problem of incompatible optimization goals and procedures, some autonomic elements have only limited interfaces for specifying goals, and much of their self-optimization is driven by built-in assumptions. For example, a self-optimizing database element may try to maximize the number of queries it can process per unit time, independent of type, yet business needs establish differential priorities among different types of query. Fortunately, such shortcomings are beginning to be recognized and addressed, notably in the area of storage [16, 48].

Finally, consider self-protection. Several general thoughts on self-protection in autonomic computing systems can be found in [38]; here I shall briefly touch upon two aspects of self-protection. First, autonomic systems must proactively protect themselves by continually monitoring all parts of the system for compliance to security policies, such as correct security settings on firewalls, sufficiently current antivirus updates, etc. and taking appropriate remedial action such as blocking network access to noncompliant components or automatically updating those components to make them compliant. A recent security offering by Cisco and IBM is one recent effort to provide some of these capabilities [26]. Second, autonomic systems will need to protect actively against perceived threats as they occur. This will require system elements that sense and handle specific types of threats within their domain of expertise to not just do their individual jobs, but to intercommunicate among themselves when they detect suspicious behavior, inducing other elements to be more vigilant, at least temporarily.

4.2 Autonomic system architectures and prototypes

An autonomic computing system must exhibit all of the self-management capabilities described in Section 4.1. In Section 3.3 I suggested that the development of competent autonomic elements requires an effective marriage of technology with architecture. Architecture is even more critical to autonomic computing systems than it is to autonomic elements. Just as a 1.3-kg pile of neurons is not a brain unless the neurons are wired together properly, an assortment of perfectly self-managing autonomic elements

will not yield an autonomic computing system unless the elements share a set of common behaviors, interfaces and interaction patterns that are demonstrably capable of engendering system-level self-management. In other words, no system-level architecture for autonomic computing is credible unless it (together with an appropriate set of technologies) can be demonstrated to support a full range of self-management capabilities in a prototype or—still better—a real deployed system.

The existing literature on multiagent systems will likely prove to be a rich source of good ideas about system-level architectures and software engineering practices [30, 31] for autonomic computing. Continued advances in the field of multiagent systems, especially as it has recently begun to recognize the importance of converging with modern trends in service-oriented computing, should be taken seriously. An early effort to develop an architecture that specifically addresses the needs of autonomic computing by marrying some of the concepts of agent-oriented and service-oriented architecture is described in reference [50]. This work explores and validates the architecture by means of an autonomic data center prototype called Unity that employs three design patterns: a self-configuration design pattern for goal-driven self assembly, a self-healing design pattern that employs sentinels and a simple cluster re-generation strategy, and a self-optimization design pattern that uses utility functions to express high-level objectives.

4.3 Autonomic system science

Several fundamental research questions pertaining to autonomic systems are discussed in [38], including the control and exploitation of emergent behavior and the potential uses of economic mechanisms such as auctions and bilateral negotiation.

To date, some of the relevant work on emergent behavior in self-managing systems has been concerned with self-organization of networks of sensors and pervasive devices [5], or peer-to-peer networks of services [45]. Other work has been concerned with system robustness and its dependence upon network structure and other properties [1, 15, 4]. In their work on Astrolabe [6] and the more recent QuickSilver project, Birman et al. have studied robust, decentralized information management based on epidemic algorithms [20]. I believe that more unified and concentrated efforts to understand and exploit emergent behavior in autonomic systems could have a radical and profound influence on their architecture, and could lead to much more decentralized and robust system structures than have yet been imagined.

The main influence of economic mechanisms upon the design of autonomic computing systems has come in two flavors. First, the idea that autonomic elements should negotiate service relationships plays a prominent role in the autonomic computing architecture of White et al. [50]—an idea that can be traced to a substantial literature on negotiation in multiagent systems. Second, Chase et al. [8, 27] and my colleagues and I at IBM [7, 49, 40] have explored the concept of using utility functions as a means for specifying high-level objectives. A more general use of economic mechanisms in autonomic computing systems may not occur for a few more years because the interactions among autonomic elements will be largely cooperative in nature. However, at the point when autonomic computing systems grow beyond the boundaries of individual organizations, issues of economic incentive will become very important, and ideas and concepts from economics will provide a rich source of inspiration for autonomic system architectures and mechanisms.

A third area of autonomic system science that is worth noting is benchmarking. Benchmarks for system performance abound: the Transaction Processing Performance Council has defined at least 4 different benchmarks, of which TPC-W (which simulates a typical web-based e-commerce environment) is perhaps the best known. The Standard Performance Evaluation Corporation (SPEC) has defined several benchmarks such as SPECjAppServer to measure the performance of web application servers. Benchmarks are useful both for quantifying and for spurring progress in the metrics that they cover. Yet benchmarks for self-management properties other than self-optimization, such as self-configuration, self-healing, and self-protection, are largely lacking. It is crucial to fill this gap. Recent notable efforts in this space include the development of dependability benchmarks [33] as well as incipient autonomic benchmarks for self-configuration and self-healing [12, 13].

5. HUMAN-COMPUTER CHALLENGES

5.1 Interfacing with humans

Autonomic computing is intended to reduce the burden of management for administrators by allowing them to express their goals and let the system take care of the details of managing to those goals. To determine how we can best support and improve their practices, it is critically important to understand how administrators manage systems today. A research team led by Paul Maglio of IBM's Almaden Research Center has been conducting ethnographic studies of system administrators, observing their behavior as they plan and rehearse complex system upgrades and diagnose problems [2]. Although this work is still in a preliminary stage, it has led to some important insights about the nature of policy that suggest that it will be a much more iterative process than many have imagined. While a simple policy such as "Update the current patch level of the operating system every 4 months" might seem plausible and easily programmable, in reality there will be all sorts of mitigating circumstances that might argue for postponement of the patch, such as concerns about software compatibility, or the system being too overloaded to bring it down for an upgrade. Supporting iterative development and flexible interpretation of policies will be a major challenge that could keep HCI researchers busy for years.

In general, there is a need to develop new languages and metaphors that will enable humans to monitor, visualize, and control autonomic systems. Humans will need to specify their goals and objectives in a natural manner, and they will need to visualize their potential effect. The techniques must be sufficiently expressive of preferences regarding cost vs. performance, security, risk and reliability, etc. Yet on the other hand they must be sufficiently structured and/or naturally suited to human psychology and cognition as to keep specification errors to a minimum—especially because high-level policies will provide a greater leverage over system behavior than traditional lower-level forms of control. In general, this is a very rich and at present little-studied field of study, and it is critical that more HCI researchers become involved.

Finally, returning to a point made in Section 4.1, another important HCI challenge for autonomic computing is to provide good intuitive interfaces that make it as easy as possible to capture expert knowledge about rules, constraints, and system models.

5.2 Policies

Policies are essential to the autonomic computing vision, as they are the form in which humans will express their objectives to autonomic systems. The good news is that there has been an enormous amount of research in the field of policy extending back many years; the bad news is that the bulk of it has been concerned with lower-level action-based policies that directly express the actions that should be taken when given conditions are satisfied. The key to the future lies in higher-level forms of policy based on goals and utility functions. Goal-based and utility-function-based policies are

much more in the spirit of autonomic computing in that they focus on the desired states, leaving it to the system to observe the current state and plan how to reach a desired state [40].

Utility-function policies, which are receiving growing attention from researchers [8, 27, 36, 49] can be viewed as extensions of goal policies. Rather than merely performing a binary classification of possible system states into "desirable" and "undesirable", they ascribe a scalar value to each possible state, so that during the course of operation the system has the freedom to select the highest-valued state. In other words, utility functions express an optimization objective—a prerequisite for true self-optimization. In principle, if objectives for self-healing, self-configuration, and/or self-protection can be defined and included in the utility function, the system can seek the feasible system state that realizes the best tradeoff among all of these properties. (Of course, this presumes that there are reasonable metrics and benchmarks for these properties).

Utility functions are an attractive candidate for a much-needed *lingua franca* for high-level objectives in autonomic computing systems. All objectives would be expressed in terms of a scalar function of service-level attributes, and these objectives could be transformed into a notion of value for lower-level resources, as has been shown in the case of resource allocation in a prototype data center [49]. However, the difficulty with utility functions is that humans find them difficult and awkward to specify. Utility functions will not be truly useful until suitable interfaces and algorithms for preference elicitation are developed for them. It may be possible to adapt preference elicitation techniques that have been developed for electronic commerce [34, 28].

6. CONCLUSIONS

Autonomic computing is a grand challenge, requiring advances in several fields of science and technology, particularly systems, software architecture and engineering, human-system interfaces, policy, modeling, optimization, and many branches of artificial intelligence such as planning, learning, knowledge representation and reasoning, multiagent systems, negotiation, and emergent behavior. Integrating these technologies and embedding them in a suitable system architecture so as to achieve the desired self-management properties is a research challenge in itself.

It is vital that, as we advance towards the ultimate vision of autonomic computing, we keep our research honest by building prototypes that quantifiably demonstrate an ever-increasing capability for self-management. In this regard, I believe it would be helpful to establish an open-platform autonomic computing prototype that would serve as a nucleus for autonomic computing researchers around the world. In the field of multiagent systems, competitions such as Robocup and the Trading Agent Competition have successfully brought researchers around the world together to focus their skill on a common problem. The TREC competitions have done the same for researchers in the field of text retrieval. However, because autonomic computing is fundamentally a *cooperative* endeavor rather than a competitive one, I envision an open-platform autonomic computing platform to which collaborators can contribute algorithms and services for others to experiment with and build from—something akin to the PlanetLab effort at www.planet-lab.org.

In summary, I hope this article serves its intended purpose of providing researchers with a broad overview of the diverse research challenges of autonomic computing and the initial efforts that many have made to address them. Frustratingly incomplete as it is, I hope this survey conveys that autonomic computing is an exciting and important application area, and I hope it will motivate researchers to lend their expertise to this growing worldwide effort. Realizing the vision of autonomic computing is necessarily a worldwide cooperative enterprise, one that will yield great societal rewards in the near-term, medium-term and long-term.

Acknowledgments

I am grateful to my many colleagues at IBM Research, particularly Steve White, Dave Chess, and Joe Hellerstein, as well as dozens of others whose names are scattered among the citations of this paper, for influencing my understanding and views of autonomic computing during the past three years. I also owe a great debt of gratitude to my fellow steering committee members Manish Parashar, Vaidy Sunderam, Rajarshi Das, Karsten Schwan, Yi-Min Wang, and Salim Hariri for working tirelessly to establish the International Conference on Autonomic Computing as a premier worldwide forum for autonomic computing research.

7. REFERENCES

[1] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 2000.

[2] R. Barrett, P. P. Maglio, E. Kandogan, and J. Bailey. Usable autonomic computing systems: the administrator's perspective. In *Proceedings of the First International Conference on Autonomic Computing*, pages 18–25. IEEE Computer Society, 2004.

[3] M. N. Bennani and D. A. Menascé. Assessing the robustness of self-managing computer systems under highly variable workloads. In *Proceedings of the First International Conference on Autonomic Computing*. IEEE Computer Society, 2004.

[4] A. Beygelzimer, G. Grinstein, R. Linkser, and I. Rish. Improving network robustness. In *Proceedings of the First International Conference on Autonomic Computing*, pages 322–323. IEEE Computer Society, 2004.

[5] K. P. Birman and R. M. Saikat Guha. Scalable, self-organizing technology for sensor networks. In B. Yeler, editor, *Advances in Pervasive Computing and Networking*. Kluwer Academic Press, 2004.

[6] K. P. Birman, R. van Renesse, and W. Vogels. Navigating in the storm: Using Astrolabe for distributed self-configuration, monitoring and adaptation. In *Proceedings of the 5th Annual International Workshop on Active Middleware Services*, 2003.

[7] C. Boutilier, R. Das, J. Kephart, G. Tesauro, and W. Walsh. Cooperative negotiation in autonomic systems using incremental utility elicitation. In *Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 89–97, 2003.

[8] R. Braynard, D. Kostic, A. Rodriguez, J. Chase, and A. Vahdat. Opus: an overlay peer utility service. In *Proceedings of the 5th International Conference on Open Architectures and Network Programming (OPENARCH)*, 2002.

[9] D. Breitgand, E. Hennis, and O. Shehory. Automated and adaptive threshold setting: Enabling technology for autonomy and self-management. In *Proceedings of the Second International Conference on Autonomic Computing*. IEEE Computer Society, 2005.

[10] M. Brodie, S. Ma, G. Lohman, T. Syeda-Mahmood, L. Mignet, N. Modani, M. Wilding, J. Champlin, and P. Sohn. Quickly finding known software problems via automated symptom matching. In *Proceedings of the Second International Conference on Autonomic Computing*. IEEE Computer Society, 2005.

[11] M. Brodie, I. Rish, S. Ma, and N. Odintsova. Active probing strategies for problem determination. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, August 2003.

[12] A. Brown, J. Hellerstein, M. Hogstrom, T. Lau, S. Lightstone, P. Shum, and M. Yost. Benchmarking autonomic capabilities: Promises and pitfalls. In *Proceedings of the First International Conference on Autonomic Computing*, pages 266–267. IEEE Computer Society, 2004.

[13] A. Brown and C. Redlin. Measuring the effectiveness of self-healing autonomic systems. In *Proceedings of the Second International*

Conference on Autonomic Computing. IEEE Computer Society, 2005.

[14] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox. Microreboot—a technique for cheap recovery. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[15] J. M. Carlson and J. Doyle. Highly optimized tolerance: A mechanism for power laws in designed systems. *Physical Review E*, 60(2):1412–1427, 1999.

[16] D. D. Chambliss, G. A. Alvarez, P. Pandey, D. Jadav, J. Xua, R. Menon, and T. P. Lee. Performance virtualization for large-scale storage systems. In *Proceedings of the 22nd International Symposium on Reliable Distributed Systems*, pages 109–118. IEEE Computer Society, 2003.

[17] D. M. Chess, G. Pacifici, M. Spreitzer, M. Steinder, A. Tantawi, and I. N. Whalley. Experience with collaborating managers: Node group manager and provisioning manager. In *Proceedings of the Second International Conference on Autonomic Computing*. IEEE Computer Society, 2005.

[18] D. M. Chess, A. Segal, I. N. Whalley, and S. R. White. Unity: Experiences with a prototype autonomic computing system. In *Proceedings of the First International Conference on Autonomic Computing*, pages 140–147. IEEE Computer Society, 2004.

[19] M. Dean and G. Schreiber. OWL web ontology language reference. http://www.w3.org/TR/owel-ref, 2004.

[20] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database management. In *Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.

[21] T. Eilam, M. Kalantar, A. Konstantinou, and G. Pacifici. Model-based automation of service deployment in a constrained environment. Technical report, IBM, 2004. RC23382.

[22] J. Gao, G. Kar, and P. Kermani. Approaches to building self healing systems using dependency analysis. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2004.

[23] Hewlett-Packard. Infrastructure and management solutions for the adaptive enterprise. Available at http://www.hp.com/products1/promos/adaptive_enterprise/pdfs/vision_for_ae.pdf.

[24] IBM. Autonomic computing: IBM's perspective on the state of information technology. Available at http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.

[25] IBM. Autonomic computing toolkit: Developer's guide. Technical Report SC30-4083-02, IBM, August 2004. Available at http://www-128.ibm.com/developerworks/autonomic/books/fpy0mst.htm.

[26] IBM. Security solutions. Available at http://www.ibm.com/security/cisco, 2005.

[27] D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing risk and reward in market-based task scheduling. In *Proceedings of the Thirteenth International Symposium on High Performance Distributed Computing (HPDC-13)*, 2004.

[28] V. Iyengar, J. Lee, and M. Campbell. Evaluating multiple attribute items using queries. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, pages 144–153, 2001.

[29] B. Jacob, R. Lanyon-Hogg, D. Nadgir, and A. Yassin. A practical guide to the IBM autonomic computing toolkit. Technical report, IBM International Technical Support Organization, 2004. http://www.redbooks.ibm.com/redbooks/SG246635.

[30] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

[31] N. R. Jennings. Building complex, distributed systems: the case for an agent-based approach. *Communications of the ACM*, 44(4):35–41, 2001.

[32] S. Kambhampati and B. Srivastava. The case for automated planning in autonomic computing. In *Proceedings of the Second International Conference on Autonomic Computing*. IEEE Computer Society, 2005.

[33] K. Kanoun, H. Madeira, and J. Arlat. A framework for dependability benchmarking. In *Proceedings of the Workshop on Dependability Benchmarking (DSN 2002)*, 2002.

[34] R. L. Keeney and H. Raiffa. *Decision with Multiple Objectives: Preferences and Value Tradeoffs*. Cambridge University Press, 1993.

[35] A. Keller, J. Hellerstein, J. Wolf, K. Wu, and V. Krishnan. The CHAMPS system: Change management with planning and scheduling. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium*. Kluwer Academic Publishers, 2004.

[36] T. Kelly. Utility-directed allocation. In *Proceedings of the First Workshop on Algorithms and Architectures for Self-Managing Systems*, 2003.

[37] J. O. Kephart, C. H. Brooks, R. Das, J. K. MacKie-Mason, R. S. Gazzale, and E. H. Durfee. Pricing information bundles in a dynamic environment. In *Proceedings of ACM EC-01*, 2001.

[38] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–52, 2003.

[39] J. O. Kephart and G. J. Tesauro. Pseudo-convergent Q-learning by competitive pricebots. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00)*, pages 463–470, Stanford, CA., 2000.

[40] J. O. Kephart and W. E. Walsh. An artificial intelligence perspective on autonomic computing policies. In *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 3–12, Los Alamitos, 2004. IEEE Computer Society.

[41] S. Koenig. Position paper: Topics for future planning competitions. In *Proceedings of the ICAPS-03 Workshop on the Competition: Impact, Organization, Evaluation, Benchmarks*, 2003.

[42] M. L. Littman, N. Ravi, E. Fenson, and R. Howard. Reinforcement learning for autonomic network repair. In *Proceedings of the First International Conference on Autonomic Computing*, pages 284–285. IEEE Computer Society, 2004.

[43] B. Melcher and B. Mitchell. Towards an autonomic framework: Self-configuring network services and developing autonomic applications. *Intel Technology Journal*, 8(4), November 2004.

[44] Microsoft. Microsoft dynamic systems initiative overview. Available at http://www.microsoft.com/windowsserversystem/dsi/dsiwp.mspx.

[45] F. Saffre and H. R. Blok. SelfService: A theoretical protocol for autonomic distribution of services in P2P communities. In *Proceedings of the First International Conference on Autonomic Computing*, pages 326–327. IEEE Computer Society, 2004.

[46] G. D. M. Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, editors. *Engineering Self-Organising Systems*. Springer-Verlag, Berlin, 2003.

[47] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, 1998.

[48] S. Uttamchandani, L. Yin, G. A. Alvarez, J. Palmer, and G. Agha. CHAMELEON: a self-evolving, fully-adaptive resource arbitrator for storage systems. In *Proceedings of Usenix*, 2005.

[49] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proceedings of the First International Conference on Autonomic Computing*, pages 70–77. IEEE Computer Society, 2004.

[50] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, A. Segal, and J. O. Kephart. Autonomic computing: Architectural approach and prototype. *Integrated Computer-Aided Engineering*, 12(2), 2005.

[51] S. Whiteson and P. Stone. Adaptive job routing and scheduling. *Engineering Applications of Artificial Intelligence Special Issue on Autonomic Computing and Automation*, 17(7):855–69, October 2004.

[52] J. Wildstrom, P. Stone, E. Witchel, R. J. Mooney, and M. Dahlin. Towards self-configuring hardware for distributed computer systems. In *The Second International Conference on Autonomic Computing*, June 2005.

[53] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox. Ensembles of models for automated diagnosis of system performance problems. Technical Report HPL-2005-3, Hewlett-Packard, January 2005.