Autonomic Computing: An Overview*

Manish Parashar[1] and Salim Hariri[2]

[1] The Applied Software Systems Laboratory,
Rutgers University, Piscataway NJ, USA
[2] High Performance Distributed Computing Laboratory,
University of Arizona, Tucson AZ, USA
parashar@caip.rutgers.edu, hariri@ece.arizona.edu

Abstract. The increasing scale complexity, heterogeneity and dynamism of networks, systems and applications have made our computational and information infrastructure brittle, unmanageable and insecure. This has necessitated the investigation of an alternate paradigm for system and application design, which is based on strategies used by biological systems to deal with similar challenges – a vision that has been referred to as autonomic computing. The overarching goal of autonomic computing is to realize computer and software systems and applications that can manage themselves in accordance with high-level guidance from humans. Meeting the grand challenges of autonomic computing requires scientific and technological advances in a wide variety of fields, as well as new software and system architectures that support the effective integration of the constituent technologies. This paper presents an introduction to autonomic computing, its challenges, and opportunities.

1 Introduction

Advances in networking and computing technology and software tools have resulted in an explosive growth in networked applications and information services that cover all aspects of our life. These sophisticated applications and services are extremely complex, heterogeneous and dynamic. Further, the underlying information infrastructure (e.g., the Internet) globally aggregates large numbers of independent computing and communication resources, data stores and sensor networks, and is itself similarly large, heterogeneous, dynamic and complex. The combination has resulted in application development, configuration and management complexities that break current computing paradigms based on static requirements, behaviors, interactions and compositions. As a result, applications, programming environments and information infrastructures are rapidly becoming brittle, unmanageable and insecure. This has necessitated the investigation of an alternate paradigm for system and application design, which is based on strategies used by biological systems to deal with similar challenges of scale, complex-

* The research presented in this paper is supported in part by the National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354 and IIS 0430826.

J.-P. Banâtre et al. (Eds.): UPP 2004, LNCS 3566, pp. 257–269, 2005.

ity, heterogeneity, and uncertainty – a vision that has been referred to as autonomic computing [5].

The *Autonomic Computing Paradigm* has been inspired by the human autonomic nervous system. Its overarching goal is to realize computer and software systems and applications that can manage themselves in accordance with high-level guidance from humans. Meeting the grand challenges of autonomic computing requires scientific and technological advances in a wide variety of fields, as well as new programming paradigm and software and system architectures that support the effective integration of the constituent technologies. This paper presents an introduction to autonomic computing, its challenges, and opportunities. In this paper, we first give an overview of the architecture of the nervous system and use it to motivate the autonomic computing paradigm. We then outline the key challenges of autonomic computing and present an overview of existing autonomic computing systems and applications.

2 The Autonomic Nervous System

The human nervous system is, to the best of our knowledge, the most sophisticated example of autonomic behavior existing in nature today. It is the body's master controller that monitors changes inside and outside the body, integrates sensory inputs, and effects appropriate response. In conjunction with the endocrine system, the nervous system is able to constantly regulate and maintain homeostasis. A homeostatic system (e.g., a large organization, an industrial firm, a cell) is an open system that maintains its structure and functions by means of a multiplicity of dynamic equilibriums that are rigorously controlled by interdependent regulation mechanisms. Such a system reacts to every change in the environment, or to every random disturbance, through a series of modifications that are equal in size and opposite in direction to those that created the disturbance. The goal of these modifications is to maintain internal balances.

The manifestation of the phenomenon of homeostasis is widespread in the human system. As an example, consider the mechanisms that maintain the concentration of glucose in the blood within limits - if the concentration should fall below about 0.06 percent, the tissues will be starved of their chief source of energy; if the concentration should rise above about 0.18 percent, other undesirable effects will occur. If the blood-glucose concentration falls below about 0.07 percent, the adrenal glands secrete adrenaline, which causes the liver to turn its stores of glycogen into glucose. This passes into the blood and the blood-glucose concentration drop is opposed. Further, a falling blood-glucose also stimulates appetite causing food intake, which after digestion provides glucose. On the other hand, if the blood-glucose concentration rises excessively, the secretion of insulin by the pancreas is increased, causing the liver to remove the excess glucose from the blood. Excess glucose is also removed by muscles and skin, and if the blood-glucose concentration exceeds 0.18 percent, the kidneys excrete excess glucose into the urine. Thus, there are five activities that counter harmful fluctuations in blood-glucose concentration [2].

The above example focuses on the maintenance of the blood-glucose concentration within safe or operational limits that have been 'predetermined' for the species. Similar control systems exist for other parameters such as systolic blood pressure, structural

integrity of the medulla oblongata, severe pressure of heat on the skin, and so on. All these parameters have a bearing on the survivability of the organism, which in this case is the human body. However, all parameters are not uniform in their urgency or their relations to lethality. Parameters that are closely linked to survival and are closely linked to each other so that marked changes in one leads sooner or later to marked changes in the others, have been termed as essential variables by Ashby in his study of the design for a brain [2]. This is discussed below.

2.1 Ashby's Ultrastable System

Every real machine embodies no less than an infinite number of variables, and for our discussion we can safely think of the human system as represented by a similar sets of variables, of which we will consider a few. In order for an organism to survive, its essential variables must be kept within viable limits (see Figure 1). Otherwise the organism faces the possibility of disintegration and/or loss of identity (i.e., dissolution or death) [14].

The body's internal mechanisms continuously work together to maintain its essential variables within their limits. Ashby's definition of adaptive behavior as demonstrated by the human body follows from this observation. He states that a form of behavior is adaptive if it maintains the essential variables within physiological limits [2] that define the viability zone. Two important observations can be made:

1. The goal of the adaptive behavior is directly linked with the survivability of the system.
2. If the external or internal environment pushes the system outside its physiological equilibrium state the system will always work towards returning to the original equilibrium state.

Ashby observed that many organisms undergo two forms of disturbances: (1) frequent small impulses to the main variables and (2) occasional step changes to its parameters. Based on this observation, he devised the architecture of the Ultra-Stable system that consists of two closed loops (see Figure 2): one that controls small disturbances and a second that is responsible for longer disturbances.

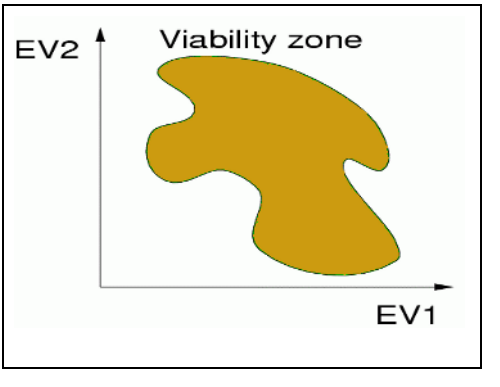

Fig. 1. Essential variables

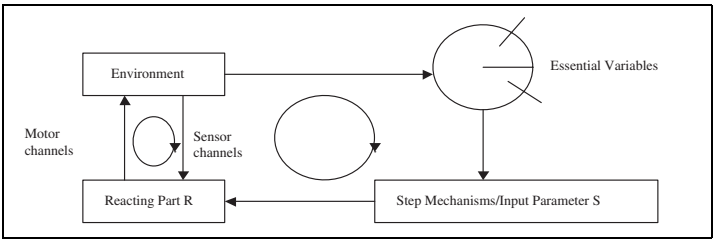

Fig. 2. The Ultra-Stable system architecture [2]

As shown in Figure 2, the ultrastable system consists of two sub-systems, the environment and the reacting part *R*. *R* represents a subsystem of the organism that is responsible for overt behavior or perception. It uses the sensor channels as part of its perception capability and motor channels to respond to the changes impacted by the environment. These set of sensors and motor channels constitute the primary feedback between *R* and the environment. We can think of *R* as a set of behaviors of the organism that gets triggered based on the changes affected by the environment. *S* represents the set of parameters that triggers changes in relevant features of this behavior set. Note that in Figure 2, *S* triggers changes only when the environment affects the essential variables in a way that causes them to go outside their physiological limits. As mentioned above, these variables need to be maintained within physiological limits for any adaptive system/organism to survive. Thus we can view this secondary feedback between the environment and *R* as responsible for triggering the adaptive behavior of the organism. When the changes impacted by the environment on the organism are large enough to throw the essential variables out of their physiological limits, the secondary feedback becomes active and changes the existing behavior sets of the organism to adapt to these new changes. Notice that any changes in the environment tend to push an otherwise stable system to an unstable state. The objective of the whole system is to maintain the subsystems (the environment and *R*) in a state of stable equilibrium. The primary feedback handles finer changes in the environment with the existing behavior sets to bring the whole system to stable equilibrium. The secondary feedback handles coarser and long-term changes in the environment by changing its existing behavior sets and eventually brings back the whole system to stable equilibrium state. Hence, in a nutshell, the environment and the organism always exist in a state of stable equilibrium and any activity of the organism is triggered to maintain this equilibrium.

2.2 The Nervous System as a Subsystem of Ashby's Ultrastable System

The human nervous system is adaptive in nature. In this section we apply the concepts underlying the Ashby's ultrastable system to the human nervous system. The nervous system is divided into the Peripheral Nervous System (PNS) and the Central Nervous System (CNS). The PNS consists of sensory neurons running from stimulus receptors that inform the CNS of the stimuli and motor neurons running from the CNS to the muscles and glands, called effectors, which take action. CNS is further divided into two parts: sensory-somatic nervous system and the autonomic nervous system. Figure 3 shows the architecture of the autonomic nervous system as an Ashby utrastable system.

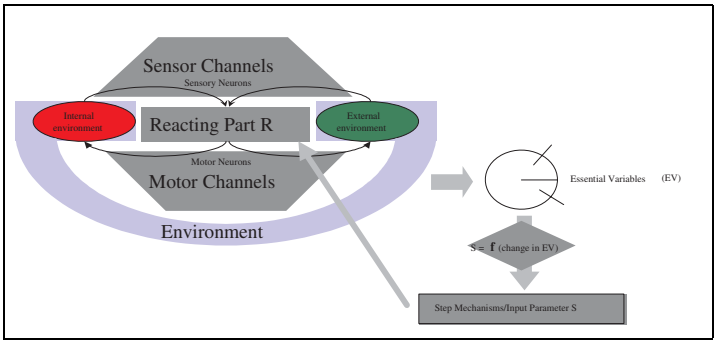

Fig. 3. Nervous system as part of an ultrastable system

As shown in Figure 3, the Sensory and Motor neurons constitute the Sensor and Motor channels of the ultrastable system. The triggering of essential variables, selection of the input parameter *S* and translation of these parameters to the reacting part R constitute the workings of the Nervous System. Revisiting the management of blood-glucose concentration within physiological limits discussed earlier, the five mechanisms that get triggered when the essential variable (i.e., concentration of glucose in blood) goes out of the physiological limits change the normal behavior of the system such that the reacting part R works to bring the essential variable back within limits. It uses its motor channels to effect changes so that the internal environment and the system (organism) come into the state of stable equilibrium. It should be noted that the environment here is divided into the internal environment and external environment. The internal environment represents changes impacted internally within the human system and the external environment represents changes impacted by the external world. However, the goal of the organism is to maintain the equilibrium of the entire system where all the sub-systems (the organism or system itself, and the internal and external environments) are in stable equilibrium.

3 The Autonomic Computing Paradigm

An autonomic computing paradigm, modeled after the autonomic nervous system, must have a mechanism whereby changes in its essential variables can trigger changes in the behavior of the computing system such that the system is brought back into equilibrium with respect to the environment. This state of stable equilibrium is a necessary condition for the survivability of the organism. In the case of an autonomic computing system, we can think of survivability as the system's ability to protect itself, recover from faults, reconfigure as required by changes in the environment, and always maintain its operations at a near optimal performance. Its equilibrium is impacted by both the internal environment (e.g., excessive memory/CPU utilization) and the external environment (e.g., protection from an external attack).

An autonomic computing system requires: (a) sensor channels to sense the changes in the internal and external environment, and (b) motor channels to react to and counter the effects of the changes in the environment by changing the system and maintaining

equilibrium. The changes sensed by the sensor channels have to be analyzed to determine if any of the essential variables has gone out of their viability limits. If so, it has to trigger some kind of planning to determine what changes to inject into the current behavior of the system such that it returns to the equilibrium state within the new environment. This planning would require knowledge to select the right behavior from a large set of possible behaviors to counter the change. Finally, the motor neurons execute the selected change. 'Sensing', 'Analyzing', 'Planning', 'Knowledge' and 'Execution' are in fact the keywords used to identify an autonomic system [7, 3]. We use these concepts to present the architecture of an autonomic element and autonomic applications and systems.

3.1 Autonomic Computing – A Holistic View

As motivated above, the emerging complexity in computing systems, services and applications requires the system/software architectures to be adaptive in all its attributes and functionality (performance, security, fault tolerance, configurability, maintainability, etc.).

We have been successful in designing and implementing specialized computing systems and applications. However, the design of general purpose dynamically programmable computing systems and applications that can address the emerging needs and requirements remains a challenge. For example, distributed (and parallel) computing has evolved and matured to provide specialized solutions to satisfy very stringent requirements in isolation, such as security, dependability, reliability, availability, performance, throughput, efficiency, pervasive/amorphous, automation, reasoning, etc. However, in the case of emerging systems and applications, the specific requirements, objectives and choice of specific solutions (algorithms, behaviors, interactions, etc.) depend on runtime state, context, and content, and are not known a priori. The goal of autonomic computing is to use appropriate solutions based on current state/context/content and on specified policies.

The computer evolution have gone through many generations starting from single process single computer system to multiple processes running on multiple geographically dispersed heterogeneous computers that could span several continents (e.g., Grid). The approaches for designing the corresponding computing systems and applications have been evolutionary and ad hoc. Initially, the designers of such systems were mainly concerned about performance, and focused intensive research on parallel processing and high performance computer architectures and applications to address this requirement. As the scale and distribution of computer systems and applications evolved, the reliability and availability of the systems and applications became the major concern. This, in turn has led to separate research in fault tolerance and reliability, and to system and applications that were ultra reliable and resilient, but not high performance. In a similar way, ultra secure computing systems and applications have been developed to meet security requirement in isolation.

This ad hoc approach has resulted in the successful design and development of specialized computing systems and applications that can optimize a few of the attributes or functionalities of computing systems and applications. However, as we highlighted before, the emerging systems and applications and their contexts are dynamic. Con-

sequently, their requirements will change during their lifetimes and may include high performance, fault tolerance, security, availability, configurability, etc. Consequently, what is needed is a new computing architecture and programming paradigm that takes a holistic approach to the design and development of computing systems and applications. Autonomic computing provides such an approach by enabling the design and development of systems/applications that can adapt themselves to meet requirements of performance, fault tolerance, reliability, security, etc., without manual intervention. Every element in an autonomic system or application consists of two main modules: the functional unit that performs the required services and functionality, and the management/control unit that monitors the state and context of the element, analyze its current requirements (performance, fault-tolerance, security, etc.) and adapts to satisfy the requirement(s).

3.2 Architecture of an Autonomic Element

An autonomic element (see Figure 4) is the smallest unit of an autonomic application or system. It is a self-contained software or system module with specified input/output interfaces and explicit context dependencies. It also has embedded mechanisms for self-management, which are responsible for implementing its functionalities, exporting constraints, managing its behavior in accordance with context and policies, and interacting with other elements. Autonomic systems and applications are constructed from autonomic elements as dynamic, opportunistic and/or ephemeral compositions. These compositions may be defined by policies and context, and may be negotiated. The key parts of an autonomic element are described below.

– **Managed Element:** This is the smallest functional unit of the application and contains the executable code (program, data structures) (e.g., numerical model of a physical process). It also exports its functional interfaces, its functional and behav-

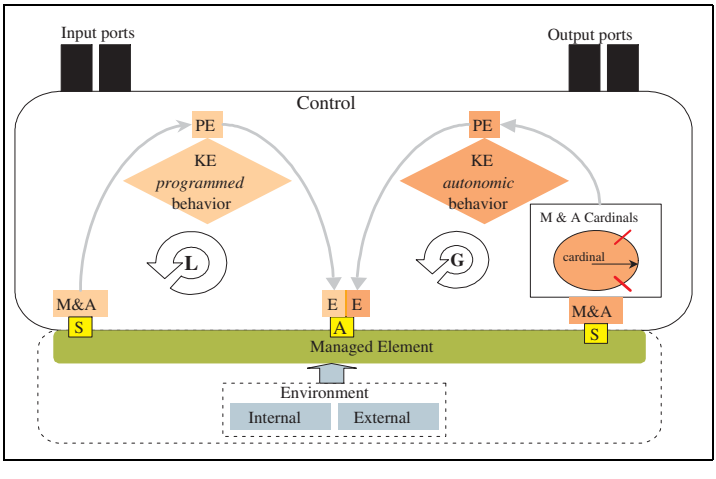

Fig. 4. An autonomic element

ioral attributes and constraints, and its control mechanisms. At runtime, the managed element can be affected in different ways, for example, it can encounter a failure, run out of resources, be externally attacked, or may hit a bottleneck impacting performance.

- **Environment:** The environment represents all the factors that can impact the managed element. The environment and the managed element can be viewed as two subsystems forming a stable system. Any change in the environment causes the whole system to go from a stable state to an unstable state. This change is then offset by reactive changes in the managed element causing the system to move back from the unstable state to a different stable state. Notice that the environment consists of two parts - internal and external. The internal environment consists of changes internal to the managed element, which reflects the state of the application/system. The external environment reflects the state of the execution environment.
- **Control:** Each autonomic element has its own manager that (1) accepts user-specified requirements (performance, fault tolerance, security, etc.), (2) interrogates the element and characterizes its state, (3) senses the state of the overall system/application, (4) determines state of the environment, and (5) uses this information to control the operation of the managed element in order to effectively achieve the specified behaviors. This control process repeats continuously throughout the lifetime of the autonomic element. As shown in Figure 4, the control part consists of two control loops - the local loop and the global loop.

  The *local loop* can only handle known environment states and is based on knowledge that is embedded in the element. Its knowledge engine contains the mapping of environment states to behaviors. For example, when the load on the local system goes above the threshold value, the local control loop will work towards balancing the load by either controlling the local resources available to the managed element or by reducing the size of the problem handled by this element. This will work only if the local resources can handle the computational requirements. However, the local loop is blind to the overall behavior of the entire application or system and thus can not achieve the desired global objectives. In a scenario where the entire system is affected, the local loop will continue repeating local optimization that may lead to degradation in performance and result in sub-optimal or chaotic behavior. At some point, one of the essential variables of the system (in this case, a performance cardinal) may overshoot their limits. This is when the global loop comes into action.

  The global loop can handle unknown environment states and may involve machine learning, artificial intelligence and/or human intervention. It uses four cardinals for the monitoring and analysis of the managed elements. These are performance, configuration, protection and security. These cardinals are like the essential variables described in Ashby's ultrastable system. This control loop results in new knowledge being introduced into the managed element to enable it to adapt its existing behaviors to respond to the changes in the environment. For example, the desired load-balancing behavior of the managed element (as directed by the local loop) requires its local load to be within prescribed limits. However, the local loop might not be able to maintain the local load within these acceptable limits, which in turn might degrade the performance of the overall system. Consequently,

this change in the overall performance cardinal triggers the global loop, which then selects an alternate behavior pattern that can address the new load conditions. The new plan is then introduced into the managed element and used to adapt its behavior.

3.3 Autonomic Computing Systems and Applications

Autonomic applications and systems are composed from autonomic elements, and are capable of managing their behaviors and their relationships with other systems/applications in accordance with high-level policies. Autonomic systems/applications exhibit eight defining characteristics [7]:

- **Self Awareness:** An autonomic application/system "knows itself" and is aware of its state and its behaviors.
- **Self Configuring:** An autonomic application/system should be able configure and reconfigure itself under varying and unpredictable conditions.
- **Self Optimizing:** An autonomic application/system should be able to detect suboptimal behaviors and optimize itself to improve its execution.
- **Self-Healing:** An autonomic application/system should be able to detect and recover from potential problems and continue to function smoothly.
- **Self Protecting:** An autonomic application/system should be capable of detecting and protecting its resources from both internal and external attack and maintaining overall system security and integrity.
- **Context Aware:** An autonomic application/system should be aware of its execution environment and be able to react to changes in the environment.
- **Open:** An autonomic application/system must function in an heterogeneous world and should be portable across multiple hardware and software architectures. Consequently it must be built on standard and open protocols and interfaces.
- **Anticipatory:** An autonomic application/system should be able to anticipate to the extent possible, its needs and behaviors and those of its context, and be able to manage itself proactively.

Sample self-managing system/application behaviors include installing software when it is detected that the software is missing (self-configuration), restarting a failed element (self-healing), adjusting current workload when an increase in capacity is observed (self-optimization) and taking resources offline if an intrusion attempt is detected (self-protecting). Each of the characteristics listed above represents an active research area. Generally, self-management is addressed in four primary system/application aspects, i.e., configuration, optimization, protection, and healing. Further, self-management solutions typically consists of the steps outlined above: (1) the application and underlying information infrastructure provide information to enable context and self awareness; (2) system/application events trigger analysis, deduction and planning using system knowledge; and (3) plans are executed using the adaptive capabilities of the application/system. An autonomic application or system implements self-managing attributes using the control loops described above to collect information, make decisions, and adapt, as necessary.

4 Autonomic Computing Research Issues and Challenges

Meeting the grand challenges of autonomic computing presents fundamental and significant research challenges that span all levels, from the conceptual level to architecture, middleware, and applications. Key research issues and challenges are presented below.

Conceptual Challenges: Conceptual research issues and challenges include (1) defining appropriate abstractions and models for specifying, understanding, controlling, and implementing autonomic behaviors; (2) adapting classical models and theories for machine learning, optimization and control to dynamic and multi agent system; (3) providing effective models for negotiation that autonomic elements can use to establish multilateral relationships among themselves; and (4) designing statistical models of large networked systems that will let autonomic elements or systems detect or predict overall problems from a stream of sensor data from individual devices.

Architecture Challenges: Autonomic applications and systems will be constructed from autonomic elements that manage their internal behavior and their relationships with other autonomic elements in accordance with policies that humans or other elements have established. As a result, system/application level self-managing behaviors will arise from the self-managing behaviors of constituent autonomic elements and their interactions. System and software architectures in which local as well as global autonomic behaviors can be specified, implemented and controlled in a robust and predictable manner remains a key research challenge.

Middleware Challenges: The primary middleware level research challenge is providing the core services required to realize autonomic behaviors in a robust, reliable and scalable manner, in spite of the dynamism and uncertainty of the system and the application. These include discovery, messaging, security, privacy, trust, etc. Autonomic systems/applications will require autonomic elements to identify themselves, discover and verify the identities of other entities of interest, dynamically establish relationships with these entities, and to interact in a secure manner. Further the middleware itself should be secure, reliable and robust against new and insidious forms of attack that use self-management based on high-level policies to their own advantage.

Application Challenges: The key challenges at the application level is the formulation and development of systems and applications that are capable of managing (i.e., configuring, adapting, optimizing, protecting, healing) themselves. This includes programming models, frameworks and middleware services that support the definition of autonomic elements, the development of autonomic applications as the dynamic and opportunistic composition of these autonomic elements, and the policy, content and context driven definition, execution and management of these applications.

5 The Autonomic Computing Landscape

There have been a number of research efforts in both academia and industry addressing autonomic computing concepts and investigating the issues outlined above. Existing

Table 1. Systems incorporating autonomic properties

System	Application area	Key autonomic issues addressed
OceanStore [4, 9]	Global, consistent, highly-available persistent data storage.	Self-healing, self-optimization, self-configuration, self-protection. Policy-based caching, routing substrate adaptation, autonomic replication, continuous monitoring, testing, and repairing.
Storage Tank [11]	Multi-platform, universally accessible storage management.	Self-optimization, self-healing. Policy-based storage and data management, server redirection and log-based recovery.
Oceano [20]	Cost effective scalable management of computing resources for software farms.	Self-optimization, self-awareness. Autonomic demands distribution, constant component monitoring.
SMART DB2 [10]	Reduction of human intervention & cost for DB2.	Self-optimization, self-configuration Autonomic index determination, disaster recovery, continuous monitoring of DB2's health and alerting the DBA.
AutoAdmin [13]	Reducing Total Cost of Ownership (TCO)	Self-tuning, self-administration. Usage tracking, index tuning and recommending based on workload.
Sabio [17]	Autonomically Classifies Large Number of documents	Self-organization, self-awareness. Group documents according to the word and phrase usage.
Q-Fabric [16]	System Support for Continuous Online Management.	Self-organization. Continuous online quality management through "customizability" of each application's QoS.

projects and products can be broadly classified as (1) systems that incorporate autonomic mechanisms for problem determination, monitoring, analysis, management, etc., into systems, and (2) systems that investigate models, programming paradigms and development environments to support the development of autonomic systems and applications. A sampling of systems belonging to these categories are summarized in Tables 1 and Table 2 respectively.

6 Summary and Conclusion

In this paper, we introduced the *autonomic computing paradigm*, which is inspired by biological systems such as the autonomic human nervous system, and enables the development of self-managing computing systems and applications. The systems/applications use autonomic strategies and algorithms to handle complexity and uncertainties with minimum human intervention. An autonomic application/system is a collection of autonomic elements, which implement intelligent control loops to

Table 2. Systems supporting development of autonomic applications and systems

System	Focus	Autonomic issues addressed
KX (Kinesthetics eXtreme) [8]	Retrofitting automicity.	Enabling autonomic properties in legacy systems.
Anthill [12]	P2P systems based on Ant colonies.	Complex adaptive behavior of P2P systems.
Astrolabe [18]	Distributed information management.	Self-configuration, monitoring and to control adaptation.
Gryphon [19]	Publish/subscribe middleware.	Large communication.
Smart Grid [21]	Autonomic principles applied to solve Grid problems.	Autonomic Grid computing.
Autonomia [6]	Model and infrastructure for enabling autonomic applications.	Autonomic applications.
AutoMate [1, 15]	Execution environment for autonomic applications.	Autonomic applications.

monitor, analyze, plan and execute using knowledge of the environment. Several research efforts focused on enabling the autonomic properties address four main areas: self-healing, self-protection, self-configuration, and self-optimization. Projects in both industry and academia, have addressed autonomic behaviors at all levels, from the hardware level to software systems and applications. At the hardware level, systems may be dynamically upgradable, while at the operating system level, active operating system code may be replaced dynamically. Efforts have also focused on autonomic middleware, programming systems and runtime. At the application level, self-optimizing databases and web servers dynamically reconfigure to adapt service performance. These efforts have demonstrated both the feasibility and promise of autonomic computing. However, achieving overall autonomic behaviors remains an open and significant challenge, which will be accomplished through a combination of process changes, skills evolution, new technologies and architecture, and open industry standards.

References

1. M. Agarwal, V. Bhat, Z. Li, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, M. Parashar, B. Khargharia, and S. Hariri. AutoMate: Enabling Autonomic Applications on the Grid. In *Proceedings of Autonomic Computing Workshop The Fifth Annual International Workshop on Active Middleware Services(AMS 2003) IEEE Computer Society Press*, pages 48–57, Seattle, WA, June 25 2003.
2. W. R. Ashby. *Design for a Brain*. Chapman & Hall Ltd, 1960.

3. IBM Corporation. An architectural blueprint for autonomic computing. April 2003.
4. UC Berkeley Computer Science Division. The OceanStore Project, Project Overview. http://oceanstore.cs.berkeley.edu/info/overview.html, July 8 2002. Project Page.
5. S. Hariri and M. Parashar. *Handbook of Bioinspired Algorithms and Applications*, chapter The Foundations of Autonomic Computing. CRC Press LLC, 2005.
6. S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri, and S. Rao. Autonomia: an autonomic computing environment. In *Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International*, April 9-11 2003.
7. P. Horn. Autonomic Computing:IBM's perspective on the State of Information Technology. http://www.research.ibm.com/autonomic/, Oct 2001. IBM Corp.
8. G. Kaiser, P. Gross, G. Kc, J. Parekh, and G. Valetto. An Approach to Autonomizing Legacy Systems. In *Workshop on Self-Healing, Adaptive and Self-MANaged Systems, SHAMAN*, New York City, NY, June 23 2002.
9. J. Kubiatowicz. OceanStore: Global-Scale Persistent Storage. http://oceanstore.cs. berkeley.edu/publications/talks/StanfordOceanStore.pdf, Spring 2001. Stanford Seminar Series, Stanford University,.
10. G. M. Lohman and S. S. Lightstone. SMART: Making DB2 (More) Autonomic. In *VLDB 2002 28th International Conference on Very Large Data Bases* , Kowloon Shangri-La Hotel, Hong Kong, China, August 20-23 2002.
11. J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg. IBM Storage Tank–A Heterogeneous Scalable SAN file system. *IBM Systems Journal*, 42(2):250–267, 2003.
12. A. Montresor. The Anthill Project Part II: The Anthill Framework. http://www.cs.unibo. it/projects/anthill/papers/anthill-4p.pdf, 2001. The Anthill Project Documentation.
13. V. Narasayya. AutoAdmin: Towards Self-Tuning Databases, November 13 2002. Guest Lecture at Stanford University.
14. University of Sussex. Adaptive system lectures. http://www.cogs.susx.ac.uk/users/ezequiel/AS/lectures/AdaptiveSystems3.ppt, 2003.
15. M. Parashar, Z. Li, H. Liu V. Matossian, and C. Schmidt. *Self-Star Properties in Complex Information Systems*, volume 3460 of *Lecture Notes in Computer Science*, chapter Enabling Autonomic Grid Applications: Requirements, Models and Infrastructures. Springer Verlag, 2005.
16. C. Poellabauer. Q-Fabric. http://www.cc.gatech.edu/systems/projects/ELinux/qfabric.html, 2002. Q-Fabric - System Support for Continuous Online Quality Management.
17. R. Pool. Natural selection. http://domino.watson.ibm.com/comm/wwwr_thinkresearch.nsf/-pages/selection200.html, 2002. A New Computer Program Classifies Documents Automatically.
18. R.V. Renesse, K.P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed systems monitoring, management, and data mining. *ACM Transaction on Computer Systems*, 21(2):164–206, 2003.
19. IBM Research. The Gryphon Project. http://www.research.ibm.com/gryphon/gryphon.html. IBM Corp.
20. IBM Research. The Océano Project. http://www.research.ibm.com/oceanoproject/. IBM Corp.
21. Columbia University Smart Grid. Smart Grid Test Bed. http://www.ldeo.columbia.edu/res/pi/4d4/testbeds/.