

Autonomic personal computing

by D. F. Bantz S. Mastrianni
 C. Bisdikian A. Mohindra
 D. Challener D. G. Shea
 J. P. Karidis M. Vanover

Autonomic personal computing is personal computing on autonomic computing platforms. Its goals combine those of personal computing with those of autonomic computing. The challenge of personal autonomic computing is to simplify and enhance the end-user experience, delighting the user by anticipating his or her needs in the face of a complex, dynamic, and uncertain environment. In this paper we identify the key technologies that enable autonomic behavior as distinguished from fault-tolerant behavior. We give some examples of current autonomic behavior and some general considerations for an architecture that supports autonomic personal computing. We identify its challenges to standards and technology developers and conclude with some guidance for future work.

Autonomic personal computing is defined here as personal computing on autonomic computing systems. It shares the goals of personal computing—responsiveness, ease of use and flexibility—with those of autonomic computing—simplicity of use, availability and security. In most cases these goals are complementary. For example, autonomic computing enhances ease of use because it eliminates or simplifies some user responsibilities. But flexibility of the location, hardware, and software configuration with personal computing complicates the job of achieving autonomic behavior. It is easier to configure, heal, optimize, and protect a system in an environment that does not change. If we can achieve autonomic behavior while still meeting the unique

needs of personal computing, millions of users will benefit worldwide.

The intention of this paper, then, is to identify the unique demands and opportunities of autonomic computing with personal devices. Our ground rules are that we seek to achieve autonomic behavior of a personal computing *system*—personal computers (PCs) and their peers, networks, and servers—not just the PC alone. We also limit our focus to application platforms, not to applications themselves. This distinction is somewhat equivocal and quantitative, however, because yesterday's applications are often tomorrow's platforms.

In what follows, we first look deeper into the meaning of the autonomic attributes of personal computing, which are different from fault-tolerant attributes. We then categorize technologies as they relate to achieving autonomic behavior in different variations: within the PC, in PC communities, and in more general systems that include servers. We give some examples of the state of the art and identify missing or incomplete capabilities. We describe some general considerations for an architecture that supports autonomic personal computing, identify some issues, and suggest a direction for future research and development.

©Copyright 2003 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

Autonomic personal computing

A computing system is autonomic if it possesses at least one of four attributes: self-configuring, self-healing, self-optimizing, and self-protecting. Autonomic personal computing exhibits and constrains these attributes in unique ways.

Self-configuring. A system is self-configuring to the extent that it automates the installation and setup of its own software in a manner responsive to the needs of the platform, the user, the peer group, and the enterprise. Personal computing often involves user-initiated configuration change, and a self-configuring system understands the implications of these changes and accommodates them automatically.

Self-healing. A system is self-healing to the extent that it monitors its own platform, detects errors or situations that may later manifest themselves as errors, and automatically initiates remediation. Fault tolerance¹ is one aspect of self-healing behavior, although the cost constraints of personal computing often preclude the redundancy required by many fault-tolerant solutions.

Self-optimizing. A system is self-optimizing to the extent that it automatically optimizes its use of its own resources. This optimization must be done with respect to criteria relevant to the needs of a specific user, his or her peer group, and the enterprise. Resource management² is one aspect of self-optimizing behavior.

Self-protecting. A system is self-protecting to the extent that it automatically configures and tunes itself to achieve security, privacy, function, and data protection goals. This behavior is of very high value to personal computing, which is exposed to insecure networks, an insecure physical environment, frequent hardware and software configuration changes, and often inadequately trained end users who may be operating under conditions of high stress. Security³ is one aspect of self-protecting behavior.

Examples of current autonomic personal computing behavior

Autonomic behavior is not new; computing systems have incorporated various forms of autonomic behavior for many years. Autonomic function creates autonomic behavior.

First, we introduce a categorization of autonomic function in terms of where it is implemented, and

then we discuss several examples that exhibit autonomic behavior in current practice.

Autonomic function can be implemented locally, drawing on locally maintained measurements and knowledge. It can be implemented among members of a peer group, sharing measurements and knowledge particular to that group. It can also be implemented using network-resident resources, in which case, measurements and knowledge are maintained for all clients. In the most general case, autonomic functions are implemented in all three ways, with different functions having a preferred implementation, resulting in the following categories:

- **Local autonomic function**—Locally, autonomic decisions can be made using knowledge that the personal computer stores or can obtain by itself, for example, from its Common Information Model (CIM)⁴ database. Local functions include automatic auditing of software configurations, local backup, surveys of the connectivity environment, and power management.
- **Peer group autonomic function**—Peer group functions require the cooperation of a local community. They include spontaneous grid computing services and knowledge sharing.
- **Network-based autonomic function**—Network-based functions enhance and extend the core autonomy of the PC. Examples include software updating, backup and restore, virus updates, and mobility support services.

Local autonomic function. Microsoft Corporation has included some local autonomic features in the Windows^{**} XP⁵ operating system, and many other autonomic features are provided by third-party utilities. Although an exhaustive review is beyond the scope of this paper, in the following subsections we describe some examples of local autonomic behavior that can be found in current practice.

Installation, configuration, and maintenance. The life cycle of a personal computer begins when it is delivered, set up, and personalized to the needs of its new user. Part of this personalization involves creating a replicable software configuration appropriate to the needs of a group of users (imaging), and part involves selectively moving the user's data and preferences from a previous platform to the new one (migration).

Technology now exists for simplifying the imaging process. In IBM's ImageUltra, a single hardware-in-

dependent super-image is created and distributed to the PC. The super-image adapts to the platform and is customized to user needs based on a key. This key, which may be specific to a particular user or common to a group, is either distributed or entered manually by the user or an administrator. A utility program transforms the super-image into the final one.

Once the system has the correct image, the user's specific information, settings, and application files need to be transferred. Windows XP includes a utility for accomplishing this transfer, with a good deal of human intervention. IBM has a comparable solution (the System Migration Agent) for Microsoft operating systems prior to Windows XP. The core logic for this product is still central to IBM's large enterprise solution for automating the migration of user content for large numbers of systems. The Ghost** product from Symantec Corporation also implements image capture, redeployment, and migration.

Change management deals with updates to system and application software after the initial installation. The Microsoft XP Automatic Update mechanism works well for updates that can be applied without special considerations, but some systems require tight policy-driven control over their configurations. The IBM Update Connector supports either user-initiated or centrally administrated updates. The CNET Networks, Inc. CatchUp is a personal Web-based service that automatically analyzes the software configuration of a system and identifies needed updates.

Break/fix primarily concerns situations in which the system was once in the correct state and needs to return to that state or one close to it. The Microsoft Windows Installer saves a valid state for the core of the operating system and for selected applications. The saved state is accessible to the user, and thus vulnerable to user error or to a security breach. IBM's solution, Rapid Restore PC, saves the complete state on a hidden partition of the hard drive of a system. Both of these solutions have only limited autonomic behavior because they require informed human intervention.

Communications. Some automation of communications tasks has become possible because of networking support services based on open standards. Using the Dynamic Host Configuration Protocol (DHCP) and domain name system (DNS) services, client devices self-configure in a network based on Transmission Control Protocol/Internet Protocol (TCP/IP).

Unfortunately, other parameters are not generally available from these services. Mail server addresses, Web proxy addresses and types, security settings such as virtual personal networks, and wireless access point settings all require manual intervention to configure today.

Further complicating the situation is the proliferation of communication technologies and standards, and the inclusion of support for multiple networks in today's mobile personal computers. Windows XP is aware of the presence or absence of each network interface, and whether or not it is connected to "active" media, that is, whether any communication is possible with the device that is one hop away. This awareness enables some autonomic behavior, for example, automatic "failover" to another network in the case of a cable fault.

Self-optimization. Windows XP Professional modifies the user interface based on the way in which the system is used. Instead of an alphabetically sorted list of programs, the user is presented with a list of the most recently used programs. Shortcuts to these programs are placed in a reserved area that allows the programs to be subsequently launched with a single click. XP also attempts to keep the desktop clean and uncluttered by consolidating the items that appear on the Windows taskbar. If multiple files are opened by the same application, the files are consolidated on the taskbar.

As another example of self-optimization, Norton Utilities** senses the current level of disk file fragmentation and alerts the user if performance might be degraded. It automatically reorganizes the physical placement of data on the disk to improve file access.

Self-protection. The user's data are a key asset and potentially vulnerable to both failures and attacks. The first line of defense is to back up those data, and many systems exist to implement backup and restore. Backup solutions in current use direct their technology toward reducing their resource usage (storage and bandwidth) as well as toward automating the initiation of the backup itself. Backup can be scheduled periodically or can be initiated in response to a hardware-initiated event, such as detection of incipient disk failure.

In the communication and storage of data, encryption is another example of a proactive protection mechanism. Data are encrypted at the time of cre-

ation, decrypted upon use, and then re-encrypted when done. Windows XP includes an encryption capability that allows selected directories and subdirectories to be encrypted. Key management remains an issue. Some IBM personal computers contain a hardware security solution, called the Embedded Security Subsystem, that generates and maintains unique encryption keys. The randomness and hardware-enforced protection of these keys supports a broad class of protection mechanisms, including digital signature, signature verification, bulk encryption, secure e-mail, and password protection.

Peer group autonomic function. Although local autonomic function depends on (hopefully) authoritative local data, and network-based autonomic function depends on data maintained by a responsible authority, peer group autonomic function depends on information solicited from other personal systems, which may or may not be accurate, timely, or relevant.

Current behavior among members of a peer group (e.g., personal computers on the same network segment) is usually confined to resource sharing—files, printers, and other peripheral devices. Here, the community of interest, the tie that binds the peer group together, is defined by the need to share information and to reduce costs by sharing expensive resources that are otherwise underutilized. Several technologies are available that allow users to discover and use resources available in the peer groups. The Microsoft Windows Browse Master allows users to discover shared folders, printers, scanners, and other resources that other Windows-based personal computers export. Systems such as Gnutella⁶ have focused primarily on enabling anonymous file-sharing among peer-group clients.

Autonomic behavior can be created on the basis of knowledge that is discovered or acquired from a peer group too. For example, some connectivity parameters (proxy server addresses) are hard to discover, but because they may be known in the peer group, they may be propagated to needy members. The community of interest for knowledge-sharing is quite specific to the type of knowledge desired. The peer group for the discovery of proxy server addresses would be, for example, all members that access the Internet.

The YouServ⁷ tool is a simple but potentially useful first step toward peer knowledge-sharing. YouServ uses existing Web technologies to achieve a very easy-

to-use system with a very low-cost implementation. Most recently, the research community has experimented with “grid computing,”⁸ a new field that focuses on large-scale resource sharing among virtual organizations. Grid technologies have focused on building protocols, services, and tools to enable virtual organizations. We believe that for a peer-group autonomic function to become a reality, technologies similar to those used in the grid need to be developed to enable virtual communities—communities of autonomic clients that collaborate to solve problems related to autonomic computing.

Network-based autonomic function. Network-based services can help PCs achieve autonomic behavior. These services can provide information to local autonomic function, provide function that complements local function or, in some cases, can even replace local function. IBM’s Access Support is an example of a service that supplies support information (e.g., BIOS [Basic Input/Output System] and device driver updates) given information about the hardware and software configuration of the PC.

Remote backup is an example of a service that complements local data protection function by providing additional storage in a physically remote location. The current generation of network-based services, specifically Internet-based services, often requires direct end-user interaction to use the service. Web services,⁹ an open standard based on the Extensible Markup Language (XML) for computer-to-computer services, obviate the necessity of human involvement, permitting the use of network-based services in a more autonomic manner.

Although some elements of a sufficient platform exist for network-based autonomic function, they have yet to be put together into a coherent partner for local or peer group autonomic personal computing. In the next two sections we explore a possible architecture that links these three types of autonomic function and the challenges that each type must meet in order to achieve seamless autonomy.

An architecture for autonomic personal computing systems

The architecture of an autonomic system, including that of a personal computing system, begins with the general architecture for autonomic systems.¹⁰ The building block of autonomic systems is depicted in Figure 1.

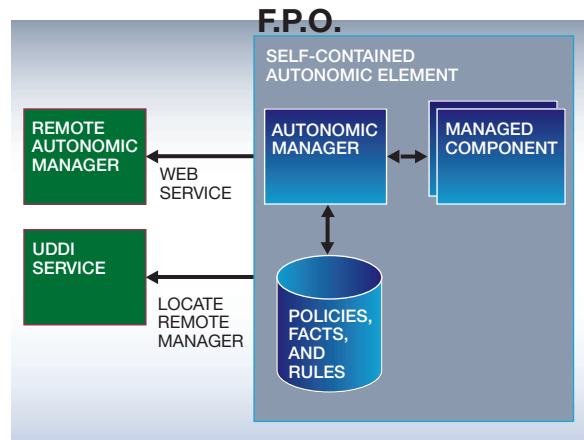
Figure 1 shows the architecture of an autonomic element (AE). Each AE consists of an autonomic manager (AM) and a set of managed components. Each managed component is responsible for communicating its events and other measurements to the local AM. In turn, based on the input received from each managed component, the AM makes decisions taking into account its policy, facts, and rules (stored locally in a database) and communicates the directives and hints to the managed component.

The figure makes a distinction between self-contained autonomy (within the large box) and autonomic management involving explicit communications with a remote manager. The interfaces between an AM and its managed component are an important part of the architecture. For Windows-based personal computing systems, one extensive set of these interfaces is represented by the Windows Management Interface,¹¹ or WMI. The interface between a remote AM and an AE is not currently standardized. This interface must be discoverable and dynamically bound so as to support self-configuration of autonomic systems; it must also be secure and private. The figure shows a remote autonomic manager implementing a Web service, located via the Universal Description, Discovery, and Integration (UDDI) service registry. We see Web service as a foundation technology because it provides standard ways to locate, communicate (via XML), compose, and interact with network-based services. But because personal systems are often mobile and occasionally disconnected, the interface must support disconnection as well.

Figure 2 shows the architecture of an autonomic system consisting of autonomic elements connected to one another at local, peer, and network levels. Resources are shown as boxes, AMs as diamond shapes, peer groups as dashed ellipses, and servers and clients as circles. Arrows represent the control exerted by AMs (e.g., S controls W). At the local level, the system consists of a single AM (e.g., A) that is capable of independent decision-making. At the peer level, each AM interacts and shares knowledge and information with its peers and may act cooperatively, as though a virtual autonomic manager (e.g., W) is present.

The notion of a virtual autonomic manager is to have the participants in a peer group achieve some level of autonomic behavior as if directed by a local or remote instance of an autonomic manager, even though none is present. The behavior is a conse-

Figure 1 Architecture of an autonomic element

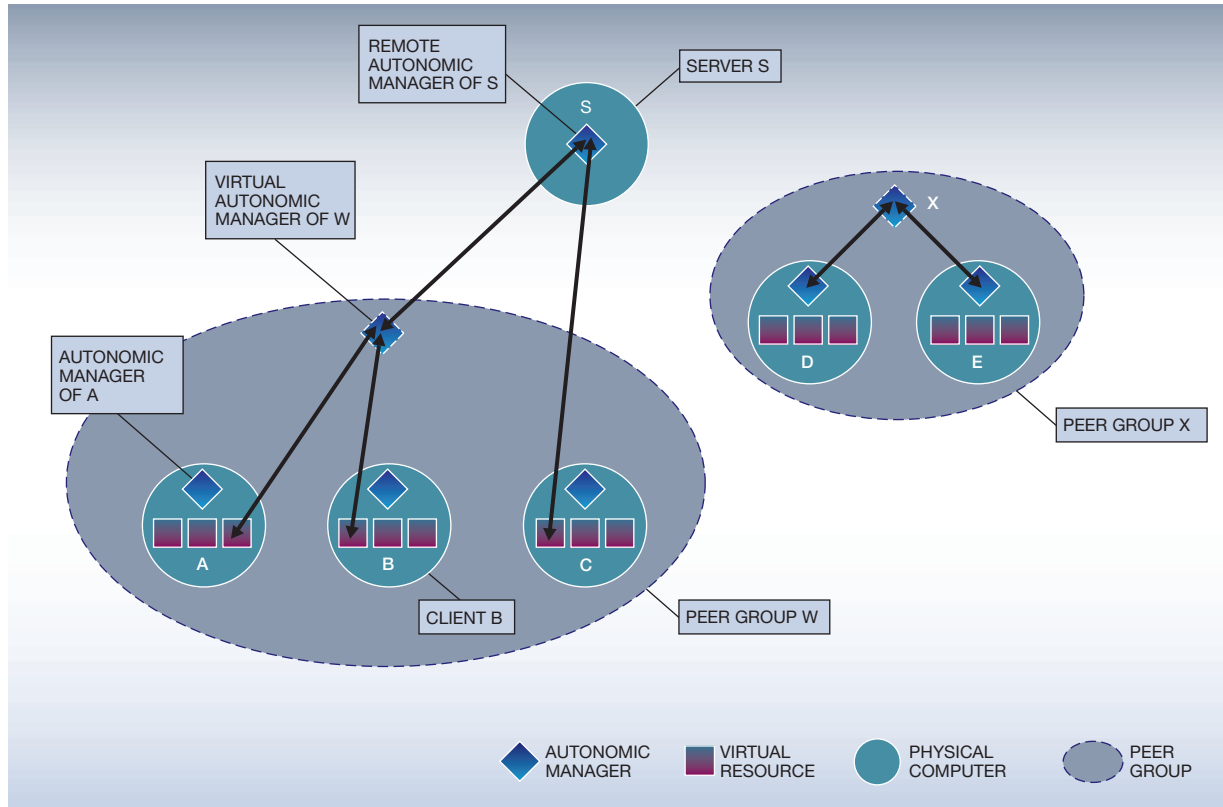


quence of peer knowledge-sharing, obtaining local consensus, and acting locally on that knowledge. The elements of a peer architecture, discovery, query, and binding, support this process.

One of the central issues of autonomic personal computing is how autonomic managers at some higher level in the system exert control over lower-level elements—what autonomic elements are controlled, how much (and how rapidly) control is exerted, and what kind of control is performed. We identify two styles of control: delegation and guidance. In delegation, a local autonomic manager passes control of some of the resources it manages to a superior. By default, the control of all local resources is delegated to the local AM. In guidance, a local autonomic manager receives information (e.g., policies) from its superior and implements them with respect to its own resources. Only one AM is ever in direct control of a resource.¹²

Today's personal computers manage all of their resources with a single manager—their operating system. It is advantageous to support a model in which control over some resources can be delegated to another manager, and this is possible with client virtualization. If we think of the boxes in Figure 2 as virtual machines, we see that two of the virtual machines of client A are under complete local control, whereas the third is under peer group control. Client C has two locally controlled virtual machines and one managed centrally, perhaps by the information technology (IT) service and support department of the enterprise.

Figure 2 An example of two hierarchies of autonomic control



In the case of occasional disconnection from a remote manager, a simple static strategy has the remote manager provide policy guidance but not real-time control. An example comes from security management. A remote, global security manager can be aware of inter- and intraenterprise issues (e.g., a new style of attack) that a disconnected client cannot discover until the critical moment of first reconnection. At that moment, a race exists between the update from the remote manager and the attack, suggesting that the client take special precautions to securely obtain the latest security guidance before resuming its normal local security policy. Such strategies support the traditional personal computing style of considerable local autonomy.

We now discuss two issues that we consider fundamental to the success of this architecture: security and privacy on the one hand, and stability on the other.

Security and privacy. Security and privacy are critical system attributes. In order for a machine to request confidential information from another machine, it needs to be able to securely identify itself. There are several ways in which this can be done, but the easiest (and probably most secure) is to use a public private key pair. The Trusted Computing Platform Alliance (TCPA)¹³ provides an ideal way of securing private keys for machine-level identification. Each system can then identify other machines either by using registered keys kept in a database, such as in the IBM Tivoli Policy director, or by using certificates.

Once identification is completed, a secure connection needs to be established between the computers. The Internet Protocol Security (IPSec)¹⁴ protocol is the standard way of making this connection, but client/server authenticated Secure Socket Layer (SSL)¹⁵ works too, as in TLS.¹⁶ This provides for au-

thenticated requests, confidential transmission of the data, and integrity of the data returned. Executables should be both signed—so that they may be verified as coming from a trusted source—and run in a sandbox.

A higher-level autonomic manager should not push information to a subject manager without it being requested. When a system is in “push” mode, the recipient needs to be able to throttle the behavior of the pushing system to avoid denial of service attacks.

In order for a machine to publish information, it needs to be able to determine whether information is confidential. Nonconfidential information must be explicitly identified; everything else is assumed to be confidential.

Stability. The goal of an autonomic personal system is to exhibit both autonomic and usable behavior to its end user. When autonomic components are put together in a system, it is not a given that the system as a whole exhibits stable behavior. For example, a personal computer may make a local evaluation that a shared communications link has high bandwidth, based on characteristics of the media, and may initiate a bandwidth-consuming activity. If other personal computers make the same decision, the link can quickly saturate and become unusable to all of them. A more conservative strategy, say, to defer bandwidth-consuming activities until a record of link performance has been established and only then to attempt more aggressive exploitation of the resource, might yield better overall behavior. In general, we imagine that constructive autonomic behavior will be constrained by context and policy. What context and which policies (and how to represent and maintain them) are important topics for future research.

Issues and directions in open autonomic personal computing

In this section we describe several challenges that available technologies do not address. Challenges exist to security, connectivity, storage, peer group collaboration, network-based services, and the user interface, and autonomic managers, virtualization, standards, and autonomic frameworks can be brought to bear to meet them.

Security. Security solutions are notorious for being difficult to use.¹⁷ Public Key Infrastructure (PKI) is theoretically a very good security design, but it is dif-

ficult to implement. Secure keys must be generated for all users and distributed to them. Certificates must be generated for all the keys, and a database of revoked certificates kept. Passwords are insecure and expensive; typically, over 50 percent of the calls to a help desk are requests to reset a password. Even biometric identification requires fingerprint templates, which must be managed. Since this management is likely to be frequent, it represents a significant security exposure.

We believe that the current lack of ease-of-use of security schemes remains a significant retardant to its widespread acceptance. There are many opportunities to apply autonomic computing technologies to security problems, among them automatic updating of security settings, secure recovery from software failures, discovery and remediation of security exposures, and the ability to manage keys securely without human intervention.

Wireless technology presents its own set of challenges, as in the widely reported problems with Wired Equivalent Privacy (WEP).¹⁸ But vulnerable security solutions are not the only source of exposures. Anyone can go to the local computer store and purchase a wireless access point, set it up in an organization, and thereby permit insecure access to the network of the organization. In the wireless arena, convenient auditing means are required, perhaps implemented as portable computers with wireless connectivity and additional instrumentation software, to detect and report these “rogue” access points and improper configuration of those installed by the organization. Wireless technology represents a new security exposure, and autonomic managers should manage its security as well.

Connectivity. There are so many alternatives to connect to personal, local, and wide area networks that this choice creates connectivity overload for users. Mobility also adds a new complexity; each locality has its own connectivity attributes. The active communication links of a device should be readjusted to the most appropriate ones each time the status of any of them changes, where a measure of appropriateness would be dependent on quality of service, cost, security, availability, location, and other policy elements. For example,

- If a device is relocated, it should automatically update its connectivity parameters to incorporate past knowledge of the new location; if it has never been in this location before, or does not know where it

is, it should try educated guesses (and ultimately, trial and error) to achieve connectivity.

- Connections should be chosen according to policies. For example, a link may be chosen if its cost per bit per second is greater than any other, subject to a minimum acceptable bandwidth. If the PC is operating on batteries and remaining power is low, the link with the lowest power may be chosen.

There is a strong interaction between autonomic behavior of connectivity and security. When switching links, active sessions may have to be migrated, exposing the session to penetration. As we share knowledge with peers, we must address the issue of what information can be shared with which peers. Personal computer file systems do provide access control per folder, but not an associated security classification, much less with access control lists that constrain what information can be shared with which peer. Although the technology exists to secure sharing, the policies to control that sharing are generally lacking.

Storage. Autonomic storage will start with automation of the storage management that users perform today. Data are often stored in multiple locations and in multiple versions, and it is too easy to lose track of where the data are located. As information is migrated and copied, significant privacy and security requirements must be met. This means that manual work, if not automated, will most likely retard the needed flow of such information.

The challenge in storage is to abstract and manage both the physical location of data and its privacy and security requirements. This abstraction should allow applications developed without autonomic storage in mind to function normally, but perhaps not as optimally as an application designed for autonomic storage. It should also provide the abstraction required for mining collaborative information. Initially, there could be some manual administration of autonomic storage through a simplified interface for setting the physical location (e.g., “add another network attached storage device”) and for setting privacy and security parameters. As autonomic storage develops, most or all of this management function should become automatic, guided by higher levels of management that implement broader business-based policy.

Peer group collaboration. The reward for peer group collaboration is access to more, and more current, data that may not be available through more formal

publishing means. Peer computing is a special case of distributed computing¹⁹ with several challenges. The first is how to form a peer group. Since autonomic function often demands implementations that do not involve human guidance, the peer group must be formed automatically. A given PC should not be constrained to be a member of only one peer group. The peer group for sharing knowledge about how to connect to the Internet from a particular place is almost certainly local to that place, whereas the peer group for sharing resources to support some grid computation need not be. It may be possible to dynamically form a peer group from a larger one (specialization) through a solicitation process in which responses to successively more specialized queries would qualify members, although the indiscriminate broadcast of the first solicitation bodes ill for systems of large scale. Limited persistence of peer groups can limit the need to solicit broadly.

A second challenge is identifying the specific collaboration type for the peer group. How can sharing be limited to just that type? How can solicitations be made both general, so that only a limited set of responses need be designed, and sharp, so that only relevant responses are generated? How are responders to generate helpful responses without compromising their own privacy?

A third challenge is determining the degree of trust that any member puts in the information obtained from any other member in the group. How can a member profit from information so obtained without complete trust in it? Consensus algorithms¹⁹ can derive plausible results, but the credibility of the responders should be taken into account. Credibility can be established through a history of acquiring useful and accurate knowledge from a member, but if no assessment of credibility is available, the uses to which obtained information can be put must be limited.

Network-based services. The opportunity for network-based services to complement and extend services implemented locally and in the peer group is too extensive to survey here, but several examples should indicate this potential.

To complement autonomic connectivity (see earlier subsection on communications), network-based services can supply additional information about resources available in the current location, such as IT resources (printers, hubs, and enhanced displays). Perhaps the key network-based service is the UDDI

service directory, from which a menu of services and how to access them can be obtained.

An analyst for International Data Corporation asserts that “The market for providing IT services is undergoing a radical change—from provisioning these services as customized offerings, generally at a customer’s site, to providing these same services from remote locations as a set of computing utility offerings . . .”²⁰ An important step in the evolution of autonomic personal computing is the development and deployment of a remote client management utility.²¹ The autonomic personal computer facilitates this utility by reducing the need for remote management to exception cases.

The utility itself consists of a secure, scalable autonomic computing infrastructure that can maintain service levels without incurring the costs of excess capacity. This infrastructure will enable capacity on demand and managed services that monitor and resolve issues before they become problems. Proactive management for problem determination, diagnosis, and resolution helps not only to reduce human involvement but also catches incipient failures early, perhaps before they precipitate cascading failures, thus reducing the overall downtime.

As client management utilities evolve, we see a continuing rebalancing of the provisioning of capabilities from the backend infrastructure, from peers, and from those resident on the client. For users to embrace a client management utility, they must be provided with end-to-end security that secures their enterprise data, coupled with sufficiently powerful remote management capabilities to enable ongoing program determination, diagnosis, and resolution without the intervention of the end users.

User interface. Many autonomic computing functions have no end-user-visible behavior.²² They implement “computing that just works.” For some users, this behavior is ideal. But for the experienced or the curious, who want to take direct control of their system parameters occasionally, it is important that they be kept up-to-date on autonomic actions that affect these parameters. How can the user be informed of just these actions? There is also the question of how the value of autonomic computing technology is perceived when the activities that deliver that value are hidden.

We believe that it is likely that autonomic personal computing will go through several stages of evolu-

tion, differentiated in part by the degree to which the end user is aware of and participates in management actions. As autonomic behavior becomes more effective, it will be trusted more, and the need for an end user to take direct control will lessen. During this time, users will likely be required to select or confirm actions that may be suggested by an autonomic manager.

Directions toward an autonomic framework. To address the issues raised in the previous subsections, we are defining an autonomic framework that brings together disparate computing elements. The key elements of the autonomic framework are the autonomic manager and the elements to be managed. The goal of the framework is to specify the interfaces and protocols for elements to exchange information and data to enable collective autonomic behavior. To achieve this goal, we need to rethink the structure of the system and the application software (and the tools that help build them) so as to identify and expose relevant and accurate indications of the state of each element, and provide standard interfaces to affect element state with minimal side effects. Each element will need an element-specific autonomic manager to monitor and control the element. This coupling of element and specific manager represents the lowest level of autonomic behavior. Elements may be isolated in virtual machines to limit undesirable interactions between them. Element-specific autonomic managers will report to a system-wide autonomic manager in a standard way. The system-wide manager is responsible for achieving end-user goals in accordance with established policy.

Autonomic managers. The elements under control of an autonomic manager must be observable and controllable. Current personal computing systems maintain a wealth of data about themselves in repositories and logs. Some of these data are redundant and confusing, and some are not even accurate. Thus the information relevant to decision-making is a challenge to obtain from those data. Similarly, many points of control exist, but their relationship to the desired behavior of the system is unclear. The job of the autonomic manager would be simplified if its input data were more indicative of root causes, and if the actions that the platform supports had a more direct effect on those causes.

Another issue in the design of autonomic managers is the representation and maintenance of knowledge:²³ relations between input data and root causes,

relations between output actions and effects, and policies constraining acceptable and desirable actions. This knowledge is often represented in rules, yet rules-based systems are notoriously hard to maintain because of the interactions between rules.

Virtualization. Virtual machines²⁴ create an efficient emulation of the environment in which operating systems run at the application protection level. This emulation permits an existing, unmodified operating system to run as an application. A virtual machine monitor allocates resources to virtual machines. Virtual machines encapsulate their contents, so that programs running within them cannot change any state outside of them.

Virtualization enhances isolation and containment, enhancing security and reducing the domain of an autonomic manager to the contents of a virtual machine. Virtualization provides finer-grained resource management and a mechanism for the capture of a complete state, so complex multiapplication execution environments can be frozen, restarted, undone, and migrated or cloned elsewhere. Virtualization provides an effective way for legacy software systems to coexist with current operating environments.

Standards. For autonomic personal computing to succeed, open standards must address the requirements of autonomic computing. Current standards activities on Resource Description Framework (RDF), Semantic Web, Simple Object Access Protocol (SOAP), UDDI, Web Services Description Language (WSDL), and Web services are focused primarily on the definition of provider-consumer or client-server interaction between entities. These standards need to be examined in the context of autonomic personal computing systems and extended to support new types of interactions, knowledge representation, and the collaboration needed to accomplish peer-to-peer autonomic behavior.

The autonomic framework that we have described is both a user of standards (e.g., Web services) and a candidate for standardization itself. As a standard it would encourage innovation in autonomic managers and the relationships between them.

Although the roles of autonomic manager and managed element are asymmetric, in peer group autonomic behavior it is often hard to statically associate the roles of a client or a server such that the current standards apply. The relationships among entities are more dynamic and reciprocal and often

evolve after discovery and negotiation. The JXTA²⁵ work is an attempt to develop a technology for generalized peer-to-peer interaction among devices.

Unfortunately, the Web services model is not symmetric; it defines a supplier and a client of the service. Initially, we expect that autonomic systems will be created through the interconnection of existing systems, augmented by local autonomic behavior. In this evolutionary paradigm system, components take pseudostatic roles as clients or servers. Eventually, however, peer roles should be supported by standards.

Personal systems are often mobile and occasionally disconnected, so any interface to a remote autonomic manager must support disconnection. Disconnection is not explicitly supported by current Web services standards. Current implementations of Web services also have a design point that is resource-intensive, rather than the resource-constrained environment of personal computing. We look forward to personal versions of application servers, databases, discovery protocols for and peer group implementations of the Web services registry, and, in general, Web services implementations compatible with the resources that are available on a single PC.

Summary and conclusions

Autonomic personal computing represents an important and distinct part of the autonomic computing concept. It is important because of the long-term potential to significantly reduce the frustration level and improve the user experience of hundreds of millions of PC users. It represents the potential to significantly reduce the personal computing support costs for millions of businesses. It is distinct because PCs exist in a more variable and less secure applications and connectivity environment. Personal computers are generally managed by less skilled personnel than are servers, centralized storage elements, or network infrastructure products.

Given the complex and dynamic environment faced by PCs (particularly mobile computers), we believe that an autonomic personal computing solution should adaptively seek out and leverage local, peer, and network resources. Sometimes the personal computing system will be disconnected from any networks and must be completely self-reliant. In other cases, the system should automatically communicate with and leverage the resources of its peers. When full network connectivity is available, the autonomic

personal computing system may choose to take advantage of services provided over the network.

Personal computing requires very careful attention to privacy and security issues. To minimize exposure to attacks and to ensure security and privacy, we believe that security approaches based solely on software are insufficient, and approaches should take advantage of tamper-resistant hardware elements (e.g., TCPA) wherever possible. Furthermore, all communication and decision-making activities related to autonomic functions for each autonomic element should be coordinated by a trusted and secure autonomic manager function associated (perhaps dynamically) with each element.

Architectures and frameworks for autonomic personal computing must be based on open standards. Autonomic personal computing should also leverage technologies and approaches such as virtualization, peer-to-peer middleware, and Web services that are currently being applied in other problem domains. But the extensive interoperability, security, and ease-of-use requirements for autonomic personal computing will require new technologies and standards to be developed.

Autonomic personal computing represents a journey rather than a destination. The journey started long ago with the introduction of features such as plug-and-play and Dynamic Host Configuration Protocol (DHCP), and it continues today with improved backup, system recovery, and network-based management tools. It will continue with further improvements to server-based monitoring and management, peer-to-peer collaboration and information sharing, and more robust stand-alone diagnostic and recovery capabilities. But the greatest progress will occur only if an open architecture for autonomic personal computing is developed to enable secure, private, transparent, and adaptive collaboration between each personal computer and its dynamic environment.

Acknowledgments

We thank those at the IBM Thomas J. Watson and Almaden Research Centers who participated in the autonomic personal computing brainstorming sessions of 2002. William Tetzlaff and Ed Lassette were responsible for the autonomic manager concept. Special thanks to Alan Ganek, Kazuo Iwano, and William Chung for their perspectives on the relationship between autonomic personal computing and

autonomic computing in general. We are indebted to the reviewers for their incisive and constructive comments.

**Trademark or registered trademark of Microsoft Corporation or Symantec Corporation.

Cited references and notes

1. D. S. Siewiorek and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, A. K. Peters Ltd. (1998).
2. *The Journal of Computer Resource Management*, Computer Measurement Group, <http://www.cmg.org>.
3. R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, John Wiley & Sons, Inc., New York (2001).
4. CIM is standardized by the Desktop Management Task Force, <http://www.dmtf.org>.
5. *Microsoft Windows XP Professional Resource Kit Documentation*, Microsoft Press, Redmond, WA (2001).
6. Gnutella, at <http://gnutella.org/>, is just one example.
7. R. J. Bayardo, R. Agrawal, D. Gruhl, and A. Somani, "YouServ: A Web-Hosting and Content Sharing Tool for the Masses," *11th International World Wide Web Conference*, Honolulu, Hawaii (May 7–11, 2002), paper available at <http://www.almaden.ibm.com/cs/people/bayardo/ps/www2002.pdf>.
8. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid—Enabling Scalable Virtual Organizations," <http://www.globus.org/research/papers.html>.
9. Web services activity of W3C, documented at <http://www.w3.org/2002/ws/>.
10. A. Ganek, "The Dawning of the Autonomic Computing Era," *IBM Systems Journal* **42**, No. 1, ●●–●● (2002, this issue).
11. Windows Management Instrumentation, Microsoft Corporation, Redmond, WA, www.microsoft.com/hwdev/driver/WMI.
12. Different AMs may be in control of independent aspects of the resource; for example, one AM may control security, whereas another controls power consumption.
13. TCPA—The Trusted Computing Platform Alliance, <http://www.trustedpc.org/>.
14. S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," RFC 2401, Internet Engineering Task Force, available at <http://www.ietf.org/rfc/rfc2401.txt>.
15. A. O. Freier, P. Karlton, and P. C. Kocher, "The SSL Protocol" (March 1996), available at <http://www.netscape.com/eng/ssl3/ssl-toc.html>.
16. T. Dierks and C. Allen, "The TLS Protocol," RFC 2246, Internet Engineering Task Force (January 1999), available at <http://www.ietf.org/rfc/rfc2246.txt>.
17. A. Whitten and J. D. Tygar, "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0," Carnegie Mellon University, Pittsburgh, PA, available at <http://www-2.cs.cmu.edu/~alma/johnny.pdf>.
18. N. Borisov, I. Goldberg, and D. Wagner, "Intercepting Mobile Communications: The Insecurity of 802.11," *Seventh International Conference on Mobile Computing and Networking*, Rome (July 2001).
19. *Distributed Systems*, S. Mullender, Editor, Addison-Wesley Publishing Co., Reading, MA (1993).
20. D. Tapper, "Is e-Sourcing IBM Global Services' Formula for Dominating the Computing Utility Market?" International Data Corporation, Framingham, MA (2001).
21. D. Bantz, A. Mohindra, and D. Shea, "The Emerging Model

- of Subscription Computing," *IT Professional* 4, No. 4, 27–32 (July/August 2002).
22. D. M. Russell, P. P. Maglio, R. Dordick, and C. Neti, "Dealing with Ghosts: Managing the User Experience of Autonomic Computing," *IBM Systems Journal* 42, No. 1, ●●–●● (2002, this issue).
 23. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Upper Saddle River, NJ (1995).
 24. R. P. Goldberg, "Survey of Virtual Machine Research," *Computer* 7, No. 6, 34–45 (1974).
 25. L. Gong, "Project JXTA: A Technology Overview," Project Juxtapose, available at <http://www.jxta.org/project/www/docs/TechOverview.pdf>.

Accepted for publication August 28, 2002.

David F. Bantz *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: bantz@watson.ibm.com)*. Dr. Bantz has been a research staff member since 1972, after a short stint at a startup company. He graduated from Columbia University in 1970 with an Eng.Sc.D. degree and taught there as an adjunct professor for nearly 25 years. He has 20 issued patents. His technical interests have always been in personal computing applications and technology, and he is currently working on autonomic personal computing.

Chatschik Bisdikian *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (electronic mail: bisdik@us.ibm.com)*. Dr. Bisdikian is a research staff member, currently working on short-range wireless networks, wireless LAN deployment, service discovery, spontaneous networking, and peer networking. He holds a Ph.D. degree in electrical engineering from the University of Connecticut. He has served as vice-chair of the IEEE 802.15.1 task group that developed a standard for personal area networks adapted from the Bluetooth specification. He is coauthor of the book *Bluetooth Revealed: The Insider's Guide to an Open Specification for Global Wireless Communications*.

David Challener *IBM Personal Computing Division, 3039 Cornwallis Road, Research Triangle Park, North Carolina 27709 (electronic mail: dchallener@vnet.ibm.com)*. Dr. Challener received his Ph.D. in applied mathematics from the University of Illinois, Champaign-Urbana and came to work for IBM upon graduation. Since then he has held positions in semiconductor yield analysis, substrate reliability, the technical staff to the president of the IBM PC Company, the Center for Natural Computing, PC architecture, and now the Personal Systems Institute, where he works on security and autonomic computing as a Senior Technical Staff Member.

John P. Karidis *IBM Personal Computing Division, Route 100, Somers, New York 10589 (electronic mail: karidis@us.ibm.com)*. Dr. Karidis is an IBM Distinguished Engineer, developing hardware and software product concepts that have included the "butterfly" keyboard on the ThinkPad® 701C, now in the permanent collection of the Museum of Modern Art in New York, and the ThinkPad TransNote portfolio computer. He received his Ph.D. in mechanical engineering from The Pennsylvania State University and joined the Watson Research Center in 1983.

Steve Mastrianni *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: stevem@us.ibm.com)*. Dr. Mastrianni is a senior software engineer currently writing autonomic software. He joined IBM Research after running a software development and consulting company for 10 years. He has authored two books, several papers, and over 70 articles, and holds a Ph.D. in computer science. He has filed over 35 patents with six issued, and he prefers writing software instead of talking about it.

Ajay Mohindra *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: ajaym@us.ibm.com)*. Dr. Mohindra has been a research staff member at IBM since 1993. He holds a Ph.D. in computer science from the Georgia Institute of Technology. His research interests include distributed systems and autonomic and e-Utility computing.

Dennis G. Shea *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: dgshea@us.ibm.com)*. Dr. Shea is the senior manager of the Personal Systems and Services department. He holds a Ph.D. in computer science from the University of Pennsylvania and electrical engineering degrees from Rensselaer Polytechnic Institute. He started his career at IBM in Boca Raton, Florida, working on small systems. His technical interests have revolved around personal systems technology, from easier connectivity and mobile solution enablement to the recent development of a desktop management computing utility.

Michael Vanover *IBM Personal Computing Division, 3039 Cornwallis Road, Research Triangle Park, North Carolina 27709 (electronic mail: vanover@us.ibm.com)*. Mr. Vanover has been a development engineer since 1986. He graduated from Wheaton College in 1982 with a B.S. in chemistry. He later obtained a master's degree in electrical and computer engineering from the University of Texas at Austin. He has nine issued patents. His technical interests have ranged from processor design to graphics and multimedia, and more recently PC manageability and security.