Editor: M. Brian Blake • mb7@georgetown.edu

# Agents and Service-Oriented Computing for Autonomic Computing

## A Research Agenda

**Frances M.T. Brazier** • *Vrije Universiteit Amsterdam*
**Jeffrey O. Kephart** • *IBM T.J. Watson Research Center*
**H. Van Dyke Parunak** • *Tech Team Government Solutions*
**Michael N. Huhns** • *University of South Carolina*

Autonomic computing is the solution proposed to cope with the complexity of today's computing environments. Self-management, an important element of autonomic computing, is also characteristic of single and multiagent systems, as well as systems based on service-oriented architectures. Combining these technologies can be profitable for all — in particular, for the development of autonomic computing systems.

In recent years, computing environments' complexity has begun to grow beyond the limits of what human system administrators can manage. This increasing complexity has three sources. First, individual components of computing systems, such as workload managers and database management systems, are becoming more difficult to configure, manage, and maintain as each release includes ever more features and tuning parameters. Second, with the advent of service-oriented computing (SOC), computing environments have become open and distributed, and components are no longer under a single organization's control. Third, and worst, the typical enterprise computing environment is a heterogeneous, irregular, multivendor pastiche that's difficult to configure, maintain, and trouble-shoot. In other words, the complexity of a modern-day computing environment is more than that of its individually complex parts.

To cope, IT vendors have recognized that there is a need for systems that assume much of their own management, referred to by many in academia and industry as *autonomic computing* systems.[1] Paul Horn, senior vice president of IBM Research, coined this term in 2001, citing an analogy with the human autonomic nervous system, which regulates heart and respiratory rates, digestion, and other bodily functions, freeing the conscious brain to focus on higher-level goals. Similarly, autonomic computing systems are expected to free system administrators to focus on higher-level goals. Autonomic computing systems can perform the following functions without human intervention:

- *self-configuration* — configuring themselves automatically when computing resources are added or removed;
- *self-healing* — discovering when, where, and why they're ailing and performing the appropriate self-repair and fault-correction operations;
- *self-optimization* — monitoring and controlling resources to ensure optimal functioning with respect to defined requirements, as well as optimizing performance and efficiency by retuning or reconfiguring themselves; and
- *self-protection* — proactively identifying and protecting themselves from arbitrary or malicious attacks or cascading failures.

Autonomic computing systems can perform these functions at both the infrastructure and

 1089-7801/09/$25.00  

application levels. The infrastructure level manages processing capacity, storage, and communication bandwidth. The application level uses the functions the infrastructure level provides. As such, autonomic computing systems strongly resemble multiagent systems (MASs). MASs, in turn, interact with services, as designed and developed within SOC. This article explores the relationships between these three paradigms: autonomic computing, MAS, and SOC.

## Integrating SOC, MAS, and Autonomic Computing

A commonly cited definition of autonomic computing systems is "computing systems that can manage themselves given high-level objectives from administrators."[1] This definition strongly resembles a commonly cited definition of a software agent, "an encapsulated computer system, situated in some environment and capable of flexible, autonomous action in that environment in order to meet its design objectives."[2] We can readily identify individual agents with individual autonomic computing elements: autonomous, adaptive entities representing resources or services that act both reactively and proactively, sensing and responding to the system environment and interacting with each other to satisfy individual goals. Thus, in both the autonomic computing and software agent paradigms, individual autonomous entities manage their own behavior, their interactions with the environment, and their interactions with other autonomous entities so as to achieve specified individual and system-wide goals. The agent paradigm includes interaction between services and agents. Although there are currently major differences between services and agents in terms of autonomy and proactiveness, our observation is that services are becoming more agent-like as they have to behave robustly and flexibly in dynamically changing execution environments. Thus, our treatment of services in this article overlaps significantly with our treatment of agents and MASs. Software agents can interact on behalf of services to negotiate service-level agreements (SLAs) across enterprise boundaries, including specifications of expected quality concerning both infrastructure and application. Most autonomic computing systems involve multiple systems, some of which will be services, and most agent systems involve multiple agents. An apt analogy is to identify autonomic computing systems with MASs.

Although the analogies among autonomic computing, services, and multiagent systems are strong and obvious, an informal survey of autonomic computing papers suggests that little transfer of ideas and technology has occurred with the MAS and SOC R&D communities. We believe that this communication failure both impoverishes the emerging autonomic computing field and deprives the agents and services communities of what could be the long-sought "killer app" — a key application that would require and inspire new developments in agent architectures and algorithms, and help multiagent systems become a mainstream, multibillion-dollar industry.

## How Agent Technology and SOC Can Benefit Autonomic Computing

Some of the earliest papers on autonomic computing[1,3] describe architectures for self-management that are strongly agent-oriented. They present a vision of autonomic computing systems composed of interacting collections of *autonomic elements* representing self-managing components such as computing resources (servers, databases, storage systems, and so on), management elements such as workload managers and monitoring systems. Each autonomic element is akin to a software agent in that it manages its own behavior by acting on data obtained from its sensors in accordance with policies and agreements established with other autonomic elements. System-level autonomic behavior arises from interactions among the autonomic elements, just as MAS behavior arises from interactions among individual software agents. These interactions are dynamic and flexible in pattern (hierarchical, peer-to-peer, and so on); relationships among agents are established via negotiation and maintained via agreements created during the negotiation process. Agreements between agents and service providers are, in fact, SLAs. Autonomic elements such as registries and sentinels play a role analogous to that of service registries and middle agents: to negotiate service provisioning as specified in the SLA.[4,5]

Let's develop this insight by discussing how the capabilities of individual agents, MAS, and SOC can contribute to autonomic computing.

### Individual Agents

Several technologies developed for individual agent systems are especially appropriate for autonomic computing, yet haven't been applied much in that realm. These include knowledge and reasoning, planning and scheduling, and interagent communication. (Learning, another key agent technology, has received considerable attention in the autonomic computing literature, so we don't discuss it in this article.)

Knowledge and reasoning are essential capabilities of a rational goal-directed agent that govern its behavior and interactions with the environment and other agents. By definition, an agent possesses knowledge of its environment, its own abilities and characteristics, and those of other agents. This knowl-

edge includes metalevel knowledge with which an agent can reason explicitly about its own beliefs, intentions, and desires as well as those of other agents. Explicit declarative metaknowledge lets an agent explicitly reason about its state and environment and determine which policies and mechanisms to use in which situations.

Self-healing, for example, requires a system to be able to recognize divergence from its normative behavior by comparing a computation model to an actual computation. Rational agents' introspective reasoning abilities are essential.

For example, in situations in which an agent's knowledge about its environment is incomplete, uncertain, or inconsistent, explicit reasoning about these aspects is warranted. All of these capabilities are largely motivated by agents' needs to interact effectively with their environment. Autonomic components have the same requirement. They're immersed in a computational environment, sensing its state and modifying it as needed, and the same techniques that enable agents to engage the external world could help autonomic components do their jobs better.

We can apply agent reasoning at both levels we discussed in the introduction: the application level needs to reason about a computation's model or requirements, whereas the infrastructure level focuses more explicitly on the nuts and bolts of the computation itself. Reasoning at this level can consider structural, functional, and behavioral characteristics:

- Structure — Are the right components involved in the right configuration? Are the right connections in place and working?
- Function — Are the inputs, outputs, preconditions, and effects (IOPE) correct? The Semantic Markup language for Web Services (OWL-S) provides a means with which function can be specified and evaluated.
- Behavior, including quality of service (QoS) — Does the computation meet its quality requirements as specified in an SLA?

Planning and scheduling let an agent determine a partial order of actions that achieves a specified goal over time, a key function for autonomic computing. Present-day planning engines' capabilities provide functionality needed by autonomic computing applications. Yet few papers and only one workshop have been devoted to applying planning to autonomic computing.[6] One stumbling block is that, to be truly practical, planners must take into account several real-world issues, such as coping with change in open environments. Specifically, a need exists for planning techniques that help assemble domain descriptions (specifications of pre- and postconditions for actions) from available data and gracefully handle incomplete domain specifications. In addition, autonomic systems must also be able to assess and plan execution progress and re-plan when plans go awry mid-course, as they inevitably will in large, complex computing environments.

Formal agent communication languages and interaction protocols govern the content and sequence of messages that agents exchange with one another. Although autonomic computing elements that are framed as Web services do respond to individual messages, those messages are relatively simple and inflexible in form, and the mapping to the core application's functionality is clear-cut. As autonomic elements begin to evolve from Web services to agents, their interactions will evolve from one-shot to extended multimessage interactions that are governed by standard interaction protocols that support negotiations or conversations. Agent toolkits will need to support: semantics and ontologies; increasingly flexible communication languages and interaction patterns; protocols to support extended, stateful interactions; and a degree of reasoning capability sufficient to drive appropriate responses to other agents' messages.

## MASs

The composition of autonomic elements into autonomic systems is strongly analogous to the composition of agents into MASs, so transitioning many multiagent paradigms and technologies to autonomic computing should be straightforward. In both paradigms, autonomic entities can negotiate contracts with other autonomic entities and other service providers for dynamic service provisioning.[5] The entities often monitor and manage the resulting agreements independently. They can form dynamic virtual organizations that manage their collective behavior in interaction with other such organizations. They might also use integration, repair, and other services provided by directories, brokers, and sentries, which themselves can be autonomous and distributed. Multiagent system research has explored many issues pertaining to multiparty service and resource negotiation. Virtual emergent organizations, auctions, and brokering are organizational structures designed for this purpose. Many different types of applications have modeled, explored, and implemented multilevel commitments between multiple agents, services, and virtual organizations, both competitive and cooperative, often specified in explicit agreements. Analogously, autonomic computing systems negotiate service contracts with (multiple) providers (either directly or through a mediator) and renegotiate such contracts when needed, taking reliability, cred-

ibility, risks, penalties, and QoS into account. Risk assessment of global system behavior with new configurations of system components or services (that is, virtual organizations) is essentially unexplored.

Markets and auctions now constitute a major subbranch of multiagent technology. Although several authors have explored using such economic mechanisms to allocate computing resources, very few of these mechanisms have been used in autonomic computing systems. These paradigms' potential is particularly strong in applications that cross enterprise boundaries.

A naïve approach to achieving self-management in the data center is to define feedback loops covering a data center's operations, comparing a current output to a predicted output and taking appropriate action if they don't match. This fails in an open environment with external services because including an uncontrollable external service inside an internally defined feedback control loop is difficult. Modern computing applications often execute in open environments that cross enterprise boundaries. Such applications include those for supply-chain management, military logistics, and e-commerce. In addition, applications based on Web services or that use the "cloud" necessarily execute in open environments. All of these application types provide important functionality for their owners and clients, and could benefit from interaction models developed in MAS.

In discussing applications of individual agent technology to autonomic computing, we noted that both agents and autonomic components need to sense, reason about, and manipulate the environment in which they're situated. This insight opens the door for one particular MAS model that might provide a path for early adoption in the autonomic systems community. This model is variously known as swarm intelligence, insect-based agents, or stigmergic systems. *Stigmergy* is a neologism from the French biologist Pierre-P. Grassé[7] to describe how social insects collaborate — that is, not by direct message exchange but by jointly making and sensing changes to a shared environment. In turn, that environment's dynamics contribute to the community's information processing. Researchers have applied principles derived from this biological model to a wide range of engineered systems.

The stigmergic or swarming model's attraction for autonomic components is that it defers the problem of adding elaborate communications protocols to existing elements. From the swarming perspective, many autonomic systems are in fact stigmergic MASs because each element both modifies and senses the shared environment, thus modulating its behavior on the basis of other elements' actions. We can apply insights, methods, and tools from the stigmergic agent community[8,9] to autonomic systems first at the analysis level, and then at a level recommending configuration changes to existing components that will yield more effective coordination among them. In this way, the autonomic community can become comfortable with concepts such as coordination and collaboration before making changes in their modules' actual design and implementation.

A few researchers[10] have built prototype autonomic computing systems that exploit the agent-like autonomy of individual components in data centers. To our knowledge, however, such systems haven't exploited agent platforms, such as the JAVA Agent Development (JADE) framework (http://jade.tilab.com)[11] or AgentScape (www.iids.org),[12] multiagent architectures such as Retsina,[13] standard agent communication languages such as the Foundation for Intelligent Physical Agent's Agent Communication Language (FIPA-ACL), or (formal) agent interaction protocols.

## SOC

One reason why agent toolkits haven't been used to build autonomic computing systems on a large scale is that such systems typically require developers to write their agents from scratch. This is a nonstarter for industry. Due to development costs and established customer bases for existing products, vendors are much more inclined to upgrade products than they are to write them afresh. The existence of service-oriented architecture (SOA) development tools such as the Eclipse SOA Tools Platform that "enable the design, configuration, assembly, deployment, monitoring, and management of software designed around a service-oriented architecture" (see www.eclipse.org/stp) make it relatively easy for vendors to use an incremental approach to create Web services from their existing products. Indeed, industry work on autonomic computing has typically framed autonomic elements as services rather than agents, and an extensive amount of standards participation has occurred

**Industry work on autonomic computing has typically framed autonomic elements as services rather than agents.**

in the context of Object Management Group (OMG), the W3C, and the Distributed Management Task Force, as opposed to FIPA.

Although the autonomic computing community's failure to use agent toolkits might seem discouraging, in fact, the SOA trend and the emergence of tools that support creating services from existing code, support two strategies for transitioning agent concepts into autonomic computing.

First, the current world of Web services that can respond to requests for service is an evolutionary step toward a world of agents (or autonomic elements) that can't just respond to such requests but can issue them as well. Services, formerly only reactive, will gradually acquire more agent capabilities as they anticipate client requests and proactively prepare for them. In other words, services will become agents. Once these agents proliferate, interacting with each other will become increasingly attractive. Second, SOA tools' emergence suggests at least two strategies for creating agent toolkits that autonomic computing developers can really use. One strategy is to create new agent toolkits that, analogous to SOA toolkits, let developers add agency to existing large, complex, commercial software rather than requiring them to develop agents from scratch. Examples of this approach have been around for some time. Two of the earliest — Hitachi's Autonomous Decentralized Control architecture (used in the control room architecture of the Shinkansen high-speed train and at Kawasaki Steel's Chiba plant)[14] and the ARCHON architecture as applied to managing an electric power grid[15,16] — made their way into the industrial world by providing agent-based wrappers for pre-existing modules.

A second strategy is to add agent functionality and communication capabilities progressively to existing SOA toolkits and enterprise service buses (ESBs), as is starting to happen with agent-enabled Web services[17] and Semantic Web services.[18]

Considerable value exists in bringing together the autonomic computing, MAS, and SOC communities. The agent community has a good deal of technology that's relevant for autonomic computing.[19] The autonomic computing community has a set of problems that are critically important to industry and potentially inspiring to agents researchers, sometimes requiring new extensions to agent research. How, then, can we forge a closer relationship between them?

First, we recommend holding cross-cultural workshops — that is, autonomic computing workshops at the major agent conferences, such as the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), and agent workshops at the major autonomic conferences, such as the International Conference on Autonomic Computing (ICAC). Specific autonomic computing challenges that are amenable to agent architecture, models, and technologies, or that require extensions to existing agent paradigms or technologies, could provide the focus. Indeed, this was the rationale for holding the 1st International Workshop on Agents for Autonomic Computing in 2008 during ICAC. Given that SOC appears to be the natural stepping stone toward agent-oriented computing, we strongly recommend involving the SOA community in these discussions, particularly those working on Semantic Web services and other SOA extensions that bring it closer to the world of agents. These and other connections between service-oriented and agent-oriented computing have been noted and discussed previously,[5] and we must capitalize on these insights. Generalizing this statement a bit, we must involve the agent-oriented software engineering and SOC communities in these workshops.

We also recommend holding an autonomic computing competition at one of the main agent or Semantic Web conferences, such as AAMAS, the Joint International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), or the International Semantic Web Conference (ISWC). Given the popularity and proven success of competitions such as RoboCup and the Trading Agent Competition in inspiring significant technical progress, we believe it would be valuable to devise a new competition that focuses the community on problems relevant to autonomic computing.

## References


1. J. Kephart and D. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, 2003, pp. 41–50.
2. N.R. Jennings, K. Sycara, and M. Wooldridge, "A Roadmap of Agent Research and Development," *J. Autonomous Agents and Multi-Agent Systems*, vol. 1, no. 1, 1998, pp. 7–38.
3. S.R. White et al., "An Architectural Approach to Autonomic Computing," *Proc. 1st Int'l Conf. Autonomic Computing* (ICAC 04), IEEE CS Press, 2004, pp. 2–9.
4. K. Decker, K. Sycara, and M. Williamson, "Middle Agents for the Internet," *Proc. 15th Int'l Joint Conf. Artificial Intelligence*, Morgan Kaufmann, 1997, pp. 578–583.
5. D.G.A. Mobach, B.J. Overeinder, and F.M.T. Brazier, "A WS-Agreement-Based Resource Negotiation Framework for Mobile Agents," *Scalable Computing: Practice and Experience*, vol. 7, no. 1, 2006, pp. 23–36.
6. B. Srivastava and S. Kambhampati, "The Case for Automated Planning in Autonomic Computing," *Proc. 2nd Int'l Conf. Autonomic Computing* (ICAC 05), IEEE CS Press, 2005, pp. 331–332.
7. P.-P. Grassé, "La Reconstruction du nid et les Coordinations Inter-Individuelles chez Bellicositermes Natalensis et Cubit-

ermes sp. La théorie de la Stigmergie: Essai d'interprétation du Comportement des Termites Constructeurs," (Reconstruction of the Nest and Coordinated Interactions among Termites According to the Theory of Stigmergy: Essays on the Interpretation of Termite Manufacturing Behavior" *Insectes Sociaux*, vol. 6, no. 1, 1959, pp. 41–84.
8. H.V.D. Parunak, "Go to the Ant: Engineering Principles from Natural Agent Systems," *Annals of Operations Research*, vol. 75, Jan. 1997, pp. 69–101.
9. H.V.D. Parunak and S.A. Brueckner, "Engineering Swarming Systems," *Methodologies and Software Eng. for Agent Systems*, F. Bergenti, M.-P. Gleizes, and F. Zambonelli, eds., Kluwer Academic Press, 2004, pp. 341–376.
10. R. Das et al., "Towards Commercialization of Utility-Based Resource Allocation," *Proc. 3rd Int'l Conf. Autonomic Computing* (ICAC 06), IEEE CS Press, 2006, pp. 287–290.
11. F. Bellifemine et al., "JADE: A White Paper," *TILAB, Torino, J.*, EXP in Search of Innovation — Special Issue on JADE, vol. 3, no. 3, 2003, pp. 6–19.
12. N.J.E. Wijngaards et al., "Supporting Internet-Scale Multi-Agent Systems," *Data and Knowledge Eng.*, vol. 41, nos. 2–3, 2002, pp. 229–245.
13. K. Sycara et al., "The RETSINA MAS Infrastructure," *Autonomous Agents and Multi-Agent Systems*, vol. 7, nos. 1–2, 2003, pp. 29–48.
14. H. Ihara and K. Mori, "Autonomous Decentralized Computer Control Systems," *Computer*, vol. 17, no. 8, 1984, pp. 57–66.
15. T. Wittig. *ARCHON: An Architecture for Multi-agent Systems*, Ellis Horwood, 1992.
16. N.R. Jennings and T. Wittig, "ARCHON: Theory and Practice," *Distributed Artificial Intelligence: Theory and Praxis*, Kluwer Academic Press, 1992.
17. L. Sheremetov and M. Contreras, "Industrial Application Integration using Agent-Enabled Semantic SO: Capnet Case Study," *Information Technology for Balanced Manufacturing Systems*, vol. 220, 2006, pp. 109–118.
18. S.A. McIlraith, T.C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, no. 2, 2001, pp. 46–53.
19. M. Huhns et al., "Research Directions for Service-Oriented Multiagent Systems," *IEEE Internet Computing*, vol. 9, no. 6, 2005, pp. 65–70.

**Frances M.T. Brazier** is a full professor at Vrije University Amsterdam, where she chairs the Intelligent Interactive Distributed Systems group. Her research focuses on many different aspects of complex adaptive systems varying from design paradigms to legal requirements to middleware support. Contact her at fmt.brazier@cs.vu.nl.

**Jeffrey O. Kephart** is the manager of the Agents and Emergent Phenomena Group at IBM T.J. Watson Research Center, where he also manages IBM's data center energy management research stategy and a joint research program with the IBM Tivoli Software Group. His research interests include autonomic computing and large-scale systems of agents. Kephart has a PhD in electrical engineering from Stanford University. Contact him at kephart@us.ibm.com.

**H. Van Dyke Parunak** is chief scientist at the Vector Research Center of TechTeam Government Solutions. His research interests include biologically inspired multiagent systems, emergent behavior in self-organizing systems, and applications of nonlinear dynamics to distributed computing. Parunak has an MA in computer and communications sciences from the University of Michigan and a PhD in near Eastern languages and civilizations from Harvard University. Contact him at van.parunak@newvectors.net.

**Michael N. Huhns** is the NCR professor of computer science and engineering at the University of South Carolina, where he also directs the Center for Information Technology. His research interests are in multiagent systems, service-oriented computing, and ontologies. Huhns has a PhD in electrical engineering from the University of Southern California. He is a fellow of the IEEE. Contact him at huhns@sc.edu.