CS886: Multi-Agent Systems

Final Project

Adjustable Autonomy for Autonomic Computing

Michael Jarrett (99318764)

July 30, 2004

Abstract

In the area of information technology infrastructures, human management is quickly becoming the largest cost of a system. Autonomic computing proposes to resolve this by creating systems that are able to configure, optimize, heal, and protect themselves.

It is unlikely that systems will ever be able to run entirely without human administrators. Adjustable autonomy will be required to allow administrators to observe and direct the system, from the highest-level goals to individual settings. Without an effective method of taking and later returning control of arbitrary components of the system, autonomic computing risks adding complexity to the system rather than reducing it.

This project describes an architecture for an autonomic computing management infrastructure in terms of a co-operative multi-agent system. An implementation technology, Cougaar, is described. Design decisions to add adjustable autonomy appropriate for both human-agent and agent-agent interactions are discussed.

1 Introduction

Moore's Law describes the trend that computers tend to double in power every eighteen months. This has allowed for computing systems to become far more powerful and intelligent than the early pioneers of computing could have ever imagined, and find their way into practically every aspect of our daily lives. However, this has not come without cost: as computers are deployed for greater number and complexity of task, the cost to administrate and maintain these systems also increase.

It has been recognized in the realm of Information Technology (IT) systems that without a change in the way such systems are managed, that the constant increase in administrative costs is unsustainable. One solution to this, proposed by IBM in 2001, is the concept of *autonomic computing*[1]. An autonomic computing system, analogous to the human nervous system, maintains the standard operation of the system. The human administrator becomes an overseer, only required for circumstances beyond the ability of the system to handle, and to specify the high-level goals of the system.

While the system would perform most maintenance on its own, the administrator still re-

quires the ability to specify the system at a high level, and in some cases, even take direct control of certain components. Even with the system in control, the administrator must still have a method of observing the system behaviour. This sort of behaviour is a textbook example of an artificial intelligence concept known as adjustable autonomy.

This project describes the adjustable autonomy requirements in an agent-based autonomic computing environment, and outlines some basic design for human-agent and agent-agent adjustable autonomy. Section 2 discusses related work, including an outline of autonomic computing and adjustable autonomy. Section 3 describes a potential design for an agent-based autonomic computing infrastructure. Section 4 discusses the requirements of, then adds the concept of adjustable autonomy to this design. Finally, section 5 presents conclusions and suggests future research initiatives.

2 Related Work

2.1 Autonomic Computing

The term autonomic computing was first coined in 2001 in a 'manifesto'[1] published by IBM. The author described the state of IT administration as 'unsustainable', and proposed a solution that would function similar to the human autonomic nervous system. Much as breathing and one's heartbeat is automatic in a human, day-to-day optimization and maintenance would be automatic in an autonomic computing system.

The biological analogy served as an effective staring point, but a more concrete definition was required. The concept was broken down by several players into four key areas of functionality a system would have to provide to be considered 'autonomic'.

- Self-Configuring: The system must be able to deploy and configure itself, and adjust this configuration when needed.
- Self-Optimizing: The system tunes itself to perform its allocated tasks more effectively.
- Self-Healing: The system can recover from most software and hardware failures with minimal disruption to normal operation.

- Self-Protecting: The system can detect and thwart malicious attempts at compromise by hackers, viruses, and other active threats to the system's operation.

In the years since, many large players from both the academic and corporate world have worked towards autonomic computing. IBM has spearheaded the research under the *autonomic computing*[2] name. HP has worked towards products under the brand *adaptive management*[3]. Microsoft has slated several technologies to be developed for its *dynamic systems initiative*[4], in conjunction with their Longhorn operating system. Sun Microsystems has introduced some autonomic concepts into its *N1* initiative, but has had trouble maintaining a consistent vision.

From the academic side, several project have been undertaken at universities. The most notable are several DARPA-funded projects under the *Dynamic Assembly of Systems for Adaptability Dependability and Assurance*[5] project. One particularly interesting project in this area is the Kinesthetics Extreme[6] project, which, like this project, used agent technology and planning to add autonomic capabilities to legacy systems.

2.2 Adjustable Autonomy

Autonomy in the context of multi-agent systems is a measure of an agent's ability and willingness to take action to accomplish its goals. This autonomy can be in relation to three key areas: a user with which it interacts, the environment in which it takes action, and other agents in the system.[7]

In most systems, agent autonomy is a design consideration for each agent, and remains fixed for the lifetime of the agent. *Adjustable autonomy* (AA) designs allow an agent's autonomy to change at runtime in response to external commands, or even autonomously.

Most of the research in adjustable autonomy focuses on human-agent interaction. The agent responsible for communicating with a user will either wait for input from the user, or take action without consulting the user. The adjustment of this autonomy level can happen based on many factors, including decisions based on a model of the user, and the relative cost of delay versus an incorrect decision. A paper on the Electric Elves project[8] looks at the issues and challenges of adjustable autonomy in a multi-agent system where agents must represent a human as well as interacting with each other. Three AA challenges are mentioned: the

coordination challenge, the team decision challenge, and the safe learning challenge.

Autonomy can be adjusted not only between a human and an agent, but also for the division of responsibilities between agents. A paper by Scerri and Reid[9] describes a design for adding AA to multi-agent systems by adding an autonomy manager to oversee and transfer autonomy between the agents under its supervision. Several of the guidelines for such a system, and more importantly, critical guidelines for agent design to support such a system are discussed.

2.3 Adjustable Autonomy in IT and AC Systems

Many sources have recognized the need for adjustable autonomy in system management, though rarely referring to it explicitly by name.

Boeing issued a whitepaper[10] describing the need for adjustable autonomy multi-agent systems in information systems, and poses several questions that they believe need to be answered to create such systems. IBM has observed that much of their work in autonomic computing will critically depend not just on system autonomy, but more importantly on the human interfaces and interactions with such a system[11]. Industry observers have noted[12] that even with the autonomic push, that automation will always eventually fail, and that once a larger portion of systems are automated, it will be even more critical for users to be able to control the system.

It was observed that even the biological analogy of autonomic systems could produce unintended side effects in humans, for example, how the"fight-or-flight" response to threats could cause undue stress in modern society. Much in the same way, user interfaces for autonomic systems must avoid such unintended side effects. IBM noted that many questions need to be answered, including "How do users understand what autonomic systems are trying to do? How should systems portray themselves to users? How can we design the experience of autonomic computing to amplify user capabilities?" [13].

The use of adjustable autonomy systems thus far in information technology infrastructure management has been uncommon thus far, because it is only with the advent of autonomic computing that such systems had a significant level of autonomy to delegate. Many major software vendors provide IT management software, but very few are equipped to automati-

cally take action, or only do so in a limited, hard-coded way. A recent article[12] observes that past automation attempts have largely failed and that only with the latest initiative into autonomic computing are we starting to see small successes.

3 Agent-Based Autonomic Computing System

While many have worked on enabling technologies, a central issue in autonomic computing has been creating the architecture for autonomic management. An approach frequently cited by IBM is that of composing autonomic systems from sets of interrelated *autonomic elements*[14]. Each of these elements consists of a feedback loop between the system being managed (the *managed element*) and the autonomic management logic (the *autonomic manager*) as shown in figure 1. The autonomic managers communicate with each other to work towards system-level goals.

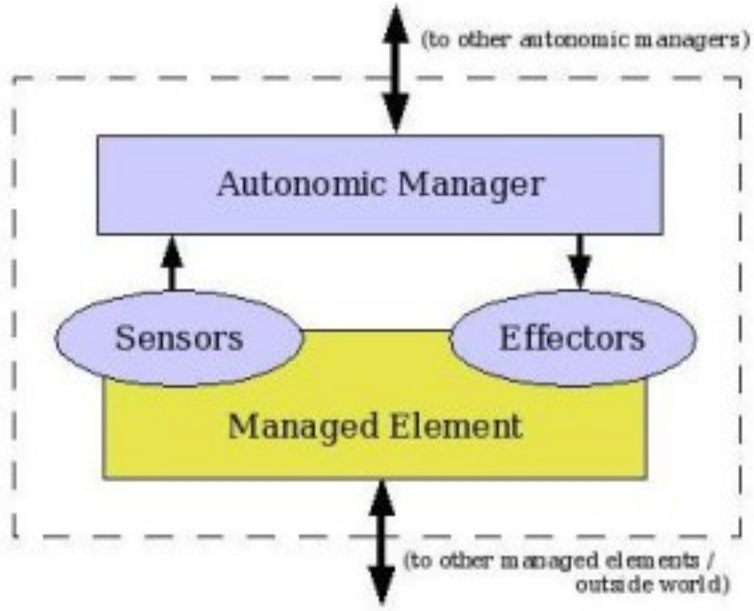

Figure 1: Autonomic Element

Proposals mentioning implementation technologies focus on the use of grid services, normally based on the Open Grid Services Architecture[15]. These systems are, at their core, simply very featureful remote procedure call libraries. An extension of the idea of grid services leads naturally to the idea of multi-agent systems, where autonomic managers are in fact intelligent agents communicating with each other.

A cooperative multi-agent system is very effective in such a situation for several reasons:

- Goal or utility driven.
- Can be given autonomy to accomplish their goals; generally can be proactive and/or reactive.
- Body of research behind cooperative negotiation and distributed planning.
- Availability of featureful multi-agent toolkits.
- Modularity and ease of upgrade.
- Can easily be used in distributed systems.

3.1 Architecture

My architecture[16], summarized in figure 2 is based on an early outline proposed[17] by IBM. The system is broken into three layers, each consisting of sets of autonomic managers. Each autonomic manager is an agent in a multi-agent system.

The lowest layer is called the *resource element* layer. Each element at this level is responsible for managing one hardware or software resource. The middle layer is called the *composite layer*, and has elements responsible for maintaining groups of similar resource elements. This layer does not handle individual management of resources, but addresses issues important to the resource as a whole, for example load distribution. At the top is the *system layer*, which is responsible for configuring and maintaining working systems. Each element operates a full system by requesting resources from the pools, and instructing each to configure itself appropriately. A system may represent either a distinct task allocated to the computing environment (eg. an e-commerce website), or a global system aspect (eg. security).

Human administrators interact with the system level by specifying the goals the system must accomplish. The system agents are responsible for recruiting lower levels of agents to perform these tasks. There is a strict hierarchy of precedence, where higher level agents ultimately have authority over lower level agents.

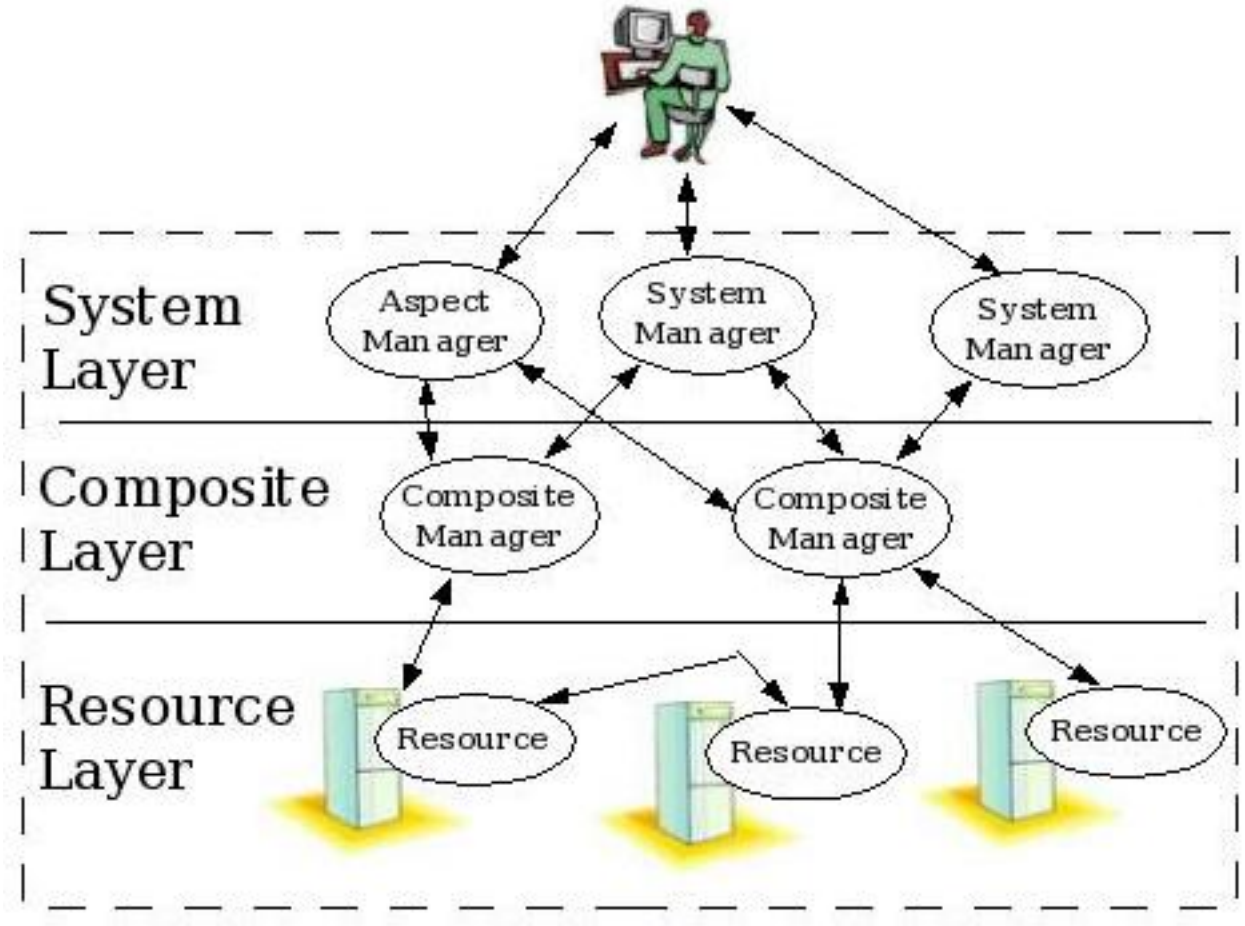

Figure 2: Layered Architecture for Autonomic Computing Management

3.2 Implementation Technology

We will construct such a system, using the Cougaar[18] multi-agent toolkit. While the architecture itself is not dependent on the use of Cougaar, the adjustable autonomy design enhancements assume a system with many of the features of Cougaar.

In Cougaar, an agent consists of a set of plugins written in Java, communicating through a shared blackboard. Through built-in communication methods, objects on an agent's blackboard may be shared with other agents in the system.

Agents may each offer a web interface through use of a Java servlet engine built into each agent. All these servlet engines are linked, allowing for communication to any agent in the system to be initiated by contacting the servlet engine of any agent.

While several 'domains' are provided by Cougaar, the main form of reasoning is that of planning. Each plugin manipulates the blackboard by publishing tasks, subtasks, assets, and the mappings between these. By modelling agents as assets, this planning system can even extend across agents in an efficient distributed manner. Tasks can have utility functions and constraints associated with them.

Additional libraries provide blackboard objects for sensor conditions, and the abilities to manipulate and aggregate these conditions. These conditions ultimately end up on the blackboards of each agent, and can be an effective way of representing an object's current state.

4 Adjustable Autonomy in Agents

It is clear in the design in section 3 that agents are not fully autonomous. Agents must give up some autonomy to form hierarchies, as well as to respond to direction from higher layers. These agents at higher layers must also have the ability to inspect the state of the agents below them to know when they themselves must take action. Central to the design of this system must be the aspect of adjustable autonomy.

4.1 Requirements

Adjustable autonomy, in the context of this autonomic computing system, requires several requirements to be solved individually in agents. The capabilities of the agents in the system determine how much autonomy is even possible, so care must be taken to meet the following requirements in the agents.

4.1.1 Observability

Each agent must have both critical portions of its state and the entirety of its current plan accessible in a form that can be interpreted by other agents. A change in autonomy can only be decided upon in an informed manner based on what information the agent presents, so by improving the quantity and quality of information presented, autonomy changes are

made more effective.

An extension of this is that agents must also make this information available in a form usable by humans. At the very minimum, a generic tool could format the data presented for other agents for display, but much more effective would be for the agent itself to present the information in a method suitable for the information domain.

The ability to view the reasoning process behind an agent's actions has been recognized[13] as a critical aspect of autonomic computing. If the agent cannot justify to an administrator the actions it has taken, the system will appear irrational to that administrator. This severely limits the ability of humans to trust the operation of the system, and could lead to administrators taking autonomy inappropriately.

4.1.2 Remote-Initiated Autonomy Transfer

Humans and other agents must have a way to take autonomy from other agents. This can encompass a variety of actions, such as forcing a particular mode of operation, barring certain actions or settings, or an order to immediately perform a particular action.

These changes must be accepted by the agent losing its autonomy without affecting the autonomy it keeps. The agent is free to (and probably should) perform its own adjustments in areas where it has kept autonomy to best suit the new constraints on its actions.

Autonomy that is taken must be able to be restored, essentially undoing the change in autonomy. An agent should seamlessly accept and utilize the autonomy returned to it.

4.1.3 Locally-Initiated Autonomy Transfer

An agent should have the ability to request more autonomy than it currently has. In a case where autonomy has been taken from it, this would look like a request to the agent taking autonomy (or a higher authority) to have some of the imposed constraints relaxed. In other cases, it may be a hardwired safety feature to always seek authorization for dangerous actions.

There may be cases where a local agent decides to give up its own autonomy. In many cases, it could simply be to finalize a period of greater autonomy where previously requested.

However, one could imagine a situation where an agent explicitly communicates to its peers that it has encountered a situation outside its own ability to handle, and is requesting intervention; essentially, an agent 'cry for help'.

One of the key issues in a locally-initiated autonomy transfer is where to request autonomy. An agent must have enough knowledge to understand which agents actually have the authority to delegate it autonomy, while not overburdening one central authority which may not have the localized knowledge to make such decisions (namely, a human administrator).

Requesting autonomy from human administrators is problematic since it requires external communication. It is likely that one or more separate agents will have to be entrusted with the knowledge to reach administrators through a variety of real-world means (pagers, emails, etc). In this case, the request can behave like a request for autonomy from another agent.

4.1.4 Persistence and Precedence

Shifts in autonomy need to be recorded, and enforced in future actions by the agent. A fully autonomous agent would immediately react to a change that deviates it from its current plan to attempt to correct the change; this sort of behaviour must be prevented. Once a particular piece of autonomy has been taken, the agent must not take any action to counteract those of the agent taking autonomy.

The reverse form of this is precedence, where autonomy is taken more than once. For example, if a composite agent takes autonomy from a resource agent, a human administrator must still be able to take that autonomy from the resource agent, also taking it from the composite agent in the process.

4.2 Implementation

We now discuss methods of implementing the requirements of adjustable autonomy for the autonomic computing management system described in section 3. We leverage Cougaar's features to meet adjustable autonomy requirements in our agent design.

4.2.1 Human Interfacing

While it is possible for a human (through an agent proxy) to manipulate the system using the same techniques as agent-agent adjustable autonomy, this would be a difficult and cumbersome way to interact with the system. An agent proxy would either need to understand the operation of every agent the human desires to control, or require the operator to communicate in the language of each agent. Neither option would present a usable interface to the end user, limiting the system's effectiveness.

We therefore leverage the fact that all Cougaar agents have a built-in Java servlet engine, allowing each agent to present a web interface. This web interface is custom for each type of agent, allowing for observation and control of that agent using controls appropriate to the particular domain.

For advanced administration, a generic plugin can be loaded into all agents, presenting a web interface for the manipulation of the agent's blackboard. This ensures that the administrator can always exert full control in cases where a custom interface lacks some required controls.

While using a web interface is effective for humans to take control, it requires an administrator to initiate contact. For agent-initiated autonomy requests, a method for agents to initiate contact with the user is required. This is handled by a single agent, the 'autonomy agent', which is treated as a proxy for the human administrators. This agent has the responsibility to determine which administrator should be contacted for a particular request, and judge the urgency with which contact is required. It must have the ability to initiate contact with administrators through means external to the system, such as pagers, emails, or warnings on a central administration console.

4.2.2 Observability

The entirety of an agent's state is stored on the blackboard[1] and is therefore observable by plugins. The agent's plan is simply the tasks, assets, and allocations between the two stored on the blackboard. The state of the managed element, including both measured conditions and enforced configuration settings, are stored as condition objects on the blackboard.

[1] This is required for agent persistence and mobility, so most Cougaar agents take special care to diligently maintain their blackboards.

Cougaar's mechanisms for condition distribution will allow these to show up on the blackboards of agents that wish to observe them. The Cougaar adaptability engine is a rule-based condition monitoring library, and can be used as an efficient way to act when conditions exceed desired parameters.

Each agent is responsible for presenting a web interface appropriate for the management of its own resource, including appropriate condition visualizations and simple control mechanisms. Since local plugins can see the entirety of the blackboard, this ensures that the web interface can also present the entirety of the plan, giving greater observability to humans than to remote agents.

The plan elements themselves are difficult for remote agents to observe in an efficient manner. This design limits observation by remote agents to condition objects only, though this is acknowledged as a potential weakness.

4.2.3 Remote-Initiated Autonomy Transfer

Remote-initiated transfers of autonomy are maintained using Cougaar *tasks*. A task represents both the statement of a goal, and constraints in achieving these goals. Handing a task to an agent is in essence taking autonomy, since the agent then is obligated to complete the task, rather than doing whatever it wants.

We modify the concept of tasks by adding priority to these tasks. The priority of tasks from agents in higher levels (eg. composite agents over resource agents) is strictly greater for higher levels, and tasks originating from a human user always have the highest priority. Agents must satisfy their highest priority tasks to the best of their ability, even if it requires reporting failure on lower priority tasks.

Tasks have both preferences and constraints. The latter serves to limit autonomy: since an agent is required to attempt to complete a task as specified, constraints limit the agent's behaviour, taking away its autonomy. Once a task specifies a constraint, the agent is forced to abide by it unless a higher priority task comes along.

This can be extended to a policy system simply by adding a 'null' task. A task which requires no further action to be taken, but has constraints serves as a policy that must be followed.

An agent is required to follow the constraints of all its tasks. Should these come into conflict,

the task with the highest priority will have its constraints satisfied, and failure reported on the other tasks.

To return autonomy, the agent taking the autonomy need only rescind the task. The constrained agent will detect this and be able to re-plan to its liking.

4.2.4 Locally-Initiated Autonomy Transfer

Agents may wish to request additional autonomy should they believe themselves to be overly constrained. Also, they could request autonomy they don't strictly need to ask for, to give an administrator the chance to review and confirm a course of action.

To request autonomy, the agent sends a 'request' task to a higher level agent. Attached to this task will be a template for the task it wishes to execute. To approve the request for additional autonomy, the agent receiving the task allocates to the requester the template task (possibly modified) with the priority of the granting agent. To reject the request, the task status is set to failed.

The autonomy can be restored to previous levels at either time by either party by rescinding the request task or the template task.

Giving up autonomy held by default is not accomplished through the task system, but implicitly through observability. An agent uses its advertised conditions to indicate to other agents that it has encountered a problem it cannot solve. In which case, an agent equipped to handle it will take autonomy from the agent having trouble. If no other agent is equipped to take the autonomy, then transferring the autonomy is not going to do much good.

5 Conclusions

It has been widely recognized that adjustable autonomy will be required to make usable autonomic computing systems. We have described a cooperative multi-agent system to satisfy the requirements of autonomic computing.

Requirements have been outlined for adjustable autonomy in the context of autonomic computing. These requirements have been largely satisfied using features available in the Cougaar

multi-agent framework. Both users and other agents can take autonomy from other agents, and give this autonomy back effectively. The autonomy taken can be very finely tuned, down to individual constraints and actions, and agents will still act autonomously in areas it has authority over.

5.1 Future Work

The most critical next step is to actually validate this design. This would involve a base implementation and real-world prototype, measured for aspects of scalability, usability, and efficacy.

Not described here are several questions about how individual agents make autonomy decisions. The possible techniques used inside individual agents need to be investigated. Requesting autonomy in particular needs to be investigated in detail. Also, the effectiveness of various approaches for determining who to request for autonomy should be investigated.

Finally, security concerns need to be addressed at the user level. The current model assumes that any administrator or agent who can access the system has complete authority will be insufficient for real-world applications. Investigations into distributing the required authentication and role knowledge will be central to this approach, as well as placing limits on who can request what autonomy from an agent.

References

[1] *Autonomic Computing: IBM's Perspective on the State of Information Technology,* IBM Research, Westchester County, N.Y., 2001.

[2] "Autonomic Computing," *IBM Research,* http://www.research.ibm.com/autonomic/ (current July 2004).

[3] "HP Press Kit: HP Adaptive Management,", *HP Newsroom: 2004 Press Kits,* http://www.hp.com/hpinfo/newsroom/press_kits/2003/amlaunch/ (current July 2004).

[4] "Dynamic Systems Initiative," *Windows Server System,* http://www.microsoft.com/windowsserversystem/dsi/default.mspx (current July 2004).

[5] "Dynamic Assembly for Systems Adaptability, Dependability, and Assurance (DASADA)," *AFRL-Rome DASASA Program Home Page,* http://www.rl.af.mil/tech/programs/dasada/ (current July 2004).

[6] "Kinesthetics eXtreme," *Programming Systems Lab,* http://www.psl.cs.columbia.edu/kx/ (current July 2004)

[7] S. Brainov and H. Hexmoor, "Quantifying Relative Autonomy in Multiagent Interation," proc. IJCAI'01 Workshop Autonomy, Delegation and Control: Interacting with Autonomous Agents, pp. 27-35, Seattle, 2001.

[8] P. Scerri, D. Pynadath and M. Tambe, "Adjustable Autonomy in Real-world Multi-Agent Environments," *proc. 5th Intl. Conf. on Autonomous Agents*, Montreal, QC, 2001.

[9] P. Scerri and N. Reed, "Designing Agents for Systems with Adjustable Autonomy," *ARTES Graduate Student Conference,* Lund, Sweden, 2001.

[10] M.S. Kerstetter and S.D.G. Smith, "Adjustable Autonomy For Human and Information System Interaction," *Autonomy, Delegation, and Control: From Inter-Agent to Groups,* tech. report WS-02-03, AAAI Press, 2002.

[11] R. Barrett, P.P. Maglio, E. Kandogan, J. Bailey, "Usable Autonomic Computing Systems: the administrator's perspective," presentation slides, http://www.mathcs.emory.edu/~vss/icac04slides/barrett-ibm.pdf (current July 2004), May 17, 2004.

[12] D. Dubie, "Net management's new autonomy," *Network World,* http://www.nwfusion.com/supp/ii2003/0224iinetman.html (current July 2004), Feb. 24, 2003.

[13] D.M. Russell et al., "Dealing with ghosts: Managing the user experience of autonomic computing," *IBM Systems Journal,* vol. 42, no. 1, 2003.

[14] S.R. White et al., "An Architectural Approach to Autonomic Computing," *proc. Intl. Conf. on Autonomic Computing,* New York, NY, 2004.

[15] "Towards Open Grid Services Architecture," *The Globus Alliance,* http://www.globus.org/ogsa/ (current July 2004).

[16] M. Jarrett, "An Architecture for Self-Protecting Autonomic Systems," tech. report, University of Waterloo, Waterloo, Ontario, 2003.

[17] *An architectural blueprint for autonomic computing,* IBM, Hawthorne, NY, 2003.

[18] "An Open Source Agent Architecture for Large-Scale Distributed Multi-Agent Systems," *COUGAAR.ORG*, http://www.cougaar.org/ (current May 2004)