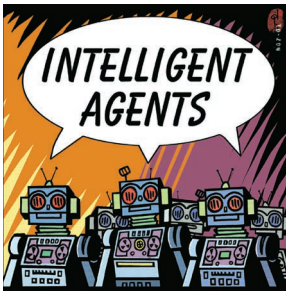
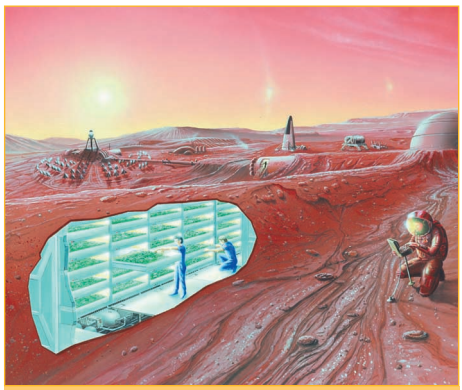
Terraforming Cyberspace



Rather than just building smarter and stronger agents, researchers must transform the wasteland of cyberspace itself, making it a safe and habitable environment for both agents and humans.

Jeffrey M. Bradshaw
Niranjan Suri
Alberto J. Cañas
Robert Davis
Kenneth Ford
Robert Hoffman
Renia Jeffers
Thomas Reichherzer
Institute for Human and Machine Cognition

During the 1940s, Jack Williamson published a series of fictional stories under the pseudonym of Will Stewart. The series described a process for attaching atmospheres to planets to make them capable of sustaining life. *Terraforming*—the term Williamson coined for this activity—first found favor with other science fiction writers, then captured the imagination of a zealous core of scientists, space advocacy groups, and other interested parties. These groups focused on Mars as the most likely target for transformation and eventual colonization. Today, many articles, books, and Web sites continue to develop the terraforming theme.


Terraforming Mars. Courtesy NASA.

Like preterraformed Mars, cyberspace currently offers a lonely, dangerous, and relatively impoverished environment for software agents. Although promoted as collaborative, agents do not easily sustain rich, long-term, peer-to-peer relationships, let alone any semblance of meaningful community involvement. Their vendors tout features that promote secure reliable interaction, but no social safety net helps agents when they get stuck or prevents them from setting the network on fire when they go off the deep end.

Despite their designers' desire to have them communicate at an almost human level, agents remain cut off from most of the world in which humans operate. Although agents are capable of self-directed mobility, severe practical restrictions limit when and where they can go. Agents are ostensibly endowed with autonomy, but the first passerby who finds the power switch can unceremoniously terminate an agent's very existence.

As a consequence of these and other limitations, most of today's agents are designed for "solitary, poor, nasty, brutish, and short" lives of narrow purpose in a relatively bounded and static computational world—their lives on the wire are just as precarious as those of Brooklyn Bridge workers a hundred years ago. With rare exception, people do not deploy today's agents in critical, long-lived, secure, or high-risk tasks, or send them on missions that require widespread agent migration or the collaboration of large numbers of agents interacting in complex, unpredictable ways.

Progress on these fronts awaits the results of ongoing research in traditional approaches to agent autonomy, collaborativity, adaptivity, and mobility. Yet we argue that focusing greater attention not only on making agents smarter and stronger but also on making the environment in which they operate more capable of sustaining various forms of agent life and civilization would simplify some of these problems. A modest terraforming effort would enable not only intelligent agents but also the agent-equivalents of dogs, insects, and chickens to survive and thrive in cyberspace.

AGENTS AND THE GRID

Fortunately, the basic infrastructure for beginning a terraforming effort is becoming more available. Designed specifically to exploit next-generation Internet capabilities, grid-based approaches provide a universal source of dynamically pluggable, pervasive, and dependable computing power, while guaranteeing levels of security and quality of service that will make new kinds of applications possible.[1] In contrast to today's static, single-purpose, and stove-piped applications, these future applications will require coordinated resource sharing and problem solving in dynamic, multi-institutional, virtual organizations. By

0018-9162/01/$10.00 © 2001 IEEE

Concern	Service	Benefit
Welfare	Social services	Get help when needed
Justice	Legal services	Get what you deserve
Environmental protection	Life-support services	Get enough to survive
Looking out for #1	Bare essentials	Get what you can take

Figure 1. Elements of terraforming for software agents. Future infrastructures must go beyond the bare essentials to provide life-support services and legal services to ensure agents' rights and help them fulfill their obligations as well as benevolent social services that offer proactive help when needed.

the time grid approaches become mainstream for large-scale applications, they will surely have migrated to ad hoc local networks of very small devices as well.

The Control of Agent-Based Systems (CoABS) agent grid (http://coabs.globalinfotek.com/) is based on Sun's Jini services. Developed at Global InfoTek with funding from the Defense Advanced Research Projects Agency, the CoABS grid provides a successful and widely used infrastructure for the large-scale integration of heterogeneous agent frameworks with object-based applications and legacy systems. Over the next few years, we expect a confluence of this effort with the efforts of the larger computational grid community (http://www.gridforum.org). The Java Agent Services Expert Group, under the auspices of Sun's Java Community Process (http://java.sun.com/aboutJava/communityprocess/jsr/jsr_087_jas.html), the OMG Agent Platform Special Interest Group (http://www.objs.com/agent/), the Foundation for Intelligent Physical Agents (http://www.fipa.org), and the FIPA Abstract Architecture Working Group (http://www.fipa.org/activities/architecture.html) are all working on essential contributions to interoperable agent infrastructure.

As Figure 1 shows, however, realizing the vision of terraforming cyberspace requires going far beyond these current efforts. Current infrastructures typically provide few safety guarantees and no incentives for agents and other components to look beyond their own selfish interests.

Brooklyn Bridge workers, 1914. Courtesy New York City Municipal Archives.

At a minimum, future infrastructures must go beyond the bare essentials to provide pervasive life-support services, which rely on mechanisms such as orthogonal persistence and strong mobility. Such services help ensure the survival of agents designed to live for many years.

Beyond the basics of individual agent protection, long-lived agent communities will depend on legal services, based on explicit policies, to ensure their rights and help them fulfill their obligations. Ultimately, benevolent social services will offer proactive help when needed.

Although some of these terraforming elements for agents exist in embryonic form within specific agent systems, the lack of underlying support at the platform level has limited their scope and effectiveness.

NOMADS LIFE-SUPPORT SERVICES

We have adopted a two-pronged approach for life-support services: enabling as much protection as possible in a standard Java virtual machine while also providing Nomads and the enhanced Aroma VM for those agent applications that require it.

For agents running in a standard JVM, we create software-based guards, which enforce policies by relying on the Java 2 security model's capabilities, including permissions and privileged code wrappers, and the Java Authentication and Authorization Service.

In contrast to other implementations of Java security, our enhanced JAAS-based approach allows revocation of access permissions under many circumstances, as well as granting different permissions to different instances of agents from the same code base. For policies that go beyond simple access-based permissions—for example, obligation policies, registration policies, conversation policies—guards implement additional auxiliary Knowledgeable Agent-oriented System (KAoS) management and enforcement capabilities as required.

Unfortunately, merely bolting new services on top of a standard JVM cannot provide some kinds of protection. Although currently the most popular and arguably the most mobility-minded and security-conscious mainstream language for agent development, Java in its current form fails to address many of the unique challenges that agent software poses. While few if any requirements for Java mobility, security, and resource management are entirely unique to agent software, nonagent software typically uses hard-coded approaches that do not allow the degree of on-demand responsiveness, configurability, extensibility, and fine-grained control that agent-based systems require.

To provide these basic life-support services, we developed Nomads, which combines Aroma—an enhanced Java-compatible virtual machine—and the Oasis agent-execution environment.[2] Nomads' current version ensures two kinds of agent environmental protection:

Resource control inhibits denial-of-service attacks and lays the foundation for QoS.

- assurance of system resources availability, even in the face of buggy agents or denial-of-service attacks; and
- protection of agent-execution state, even in the face of unanticipated system failure.

For agents running in the Aroma VM, we can create a considerably more powerful guarded environment that provides not only the standard Java and KAoS enforcement capabilities, but also supports access revocation under all circumstances, as well as dynamic resource control and full-state capture on demand for any Java agent or service.

Protecting agent and system resources

Full appreciation of Aroma and Nomads resource-control features requires some understanding of the current Java security model. Early versions of Java relied on the sandbox model to protect mobile code from accessing dangerous methods. In contrast, Java 2 uses a permission-based security model. Unlike the previous all-or-nothing approach, Java applets and applications can now have varying degrees of access to system resources.

Unfortunately, current Java mechanisms do not address the problem of resource control. While it may be possible to prevent a Java program from writing to any directory except /tmp, once the program is given permission to write to the /tmp directory, it places no further restrictions on the program's I/O. In the current Java implementation, there is no way to limit the amount of disk space the program can use or to control the rate at which the program reads and writes from the network.

Resource control is important for several reasons:

- Without it, systems and networks are open to denial-of-service attacks through resource overuse.
- It lays the foundation for quality-of-service guarantees. To make any such guarantees about the availability of resources for a given task, the system must be able to limit resource usage by other tasks—a capability the current Java environment lacks.
- Resource control presupposes resource accounting, which allows measuring the resources the overall system or some component of it consumes for either billing or monitoring purposes. Monitoring resource utilization over time allows the system to detect abnormal behavior.
- The availability of resource control mechanisms in the environment simplifies the task of developing systems for resource-constrained situations.

Aroma currently provides a comprehensive set of CPU, disk, and network resource controls. These mechanisms allow the system to place limits on both the rate and quantity of resources the Java threads use. Rate limits include CPU usage, disk read rate, disk write rate, network read rate, and network write rate. The system specifies I/O rate limits in bytes per millisecond. Quantity limits include disk space, total bytes written to disk, total bytes read from the disk, total bytes written to the network, and total bytes read from the network. Because the system provides resource controls at the computing environment level, the controls are completely transparent to the executing programs, requiring no modification to the Java code.

In agent environments, several uses of the Nomads-based resource control mechanisms are possible. First, the KAoS domain manager and the VM-level guard use the resource control mechanism to place limits on the resources that services and components running within the Aroma VM consume. The guard can vary the resource limits to accommodate changes in policy affecting level-of-service guarantees. The guard also takes advantage of the resource accounting capabilities to measure and report on the resources that services and components consume and, if policy permits, to look for patterns of resource abuse that might signal denial-of-service attacks, taking autonomous action to reduce the resources available to the attacker accordingly.

Protecting agent state

Protecting agent state requires saving the running agent or component's entire state, including its execution stack, so that it can be fully restored in case of system failure or a need to temporarily suspend its execution. The standard term for this process is *checkpointing*. Over the past few years, researchers at Sun Microsystems and elsewhere have also developed the more general concept of *transparent persistence*, sometimes called *orthogonal persistence*. This research seeks to define language-independent principles and language-specific mechanisms for making persistence available for all data, irrespective of type. Ideally, the approach would not require any special work by the programmer—for example, implementing serialization methods in Java or using transaction interfaces in conjunction with object databases—and there would be no distinction between short-lived and long-lived data.

We have used the Nomads state capture features extensively for agents that require anytime mobility, whether to perform a task or to allow immediate escape from a host under attack or about to go down, a scenario we call "scram." For example, as part of the DARPA UltraLog program, we are using mobile state to improve the survivability of agent systems in

the face of information warfare and kinetic attacks. We are also using this feature for transparent load-balancing and forced-code migration on demand in distributed computing applications.[3] To support transparent persistence for agents and other distributed-system components, we are implementing scheduled and on-demand checkpointing services that will preserve agent execution state, even in the face of unanticipated system failure.

While others have implemented many of these life-support features in limited ways for specific applications, we are working with academic and industrial research partners to make their use more routine and pervasive so that developers can rely on them as a matter of course. As our understanding of requirements for basic agent life-support services increases, we are beginning to anticipate what additional layers of legal and social services might begin to look like.

KAOS LEGAL AND SOCIAL SERVICES

Terraforming cyberspace involves more than regulating computing resources and protecting agent state. As the scale and sophistication of agents grow and their lifespan increases, agent developers and users will want the ability to express complex high-level constraints on agent behavior within a given environment. It seems inevitable that productive interaction between agents in long-lived communities will also require some kind of legal services, based on explicit enforceable policies, to ensure their rights and help them fulfill their obligations. Over time, it seems likely that benevolent social services will also eventually evolve to offer help with individual agent or systemic problems.

In both legal and social service arenas, it is clear that preventive initiatives are usually superior to after-the-fact remedies. As in the well-known poem by Joseph Malins, we liken the former to a "fence at the top of the cliff" and the latter to an "ambulance down in the valley."

We base our approach on the assumption that preventive policy-based *fences* can complement and enhance after-the-fact remedial *ambulance-in-the-valley* mechanisms. The policies governing some set of agents should describe expected behavior in sufficient detail to allow easily anticipating or detecting deviations. At the same time, related policy support services help make compliance as easy as possible.

Complementing these policy support services, various enforcement mechanisms operate as a sort of "cop at the top of the cliff" to warn of potential problems before they occur. When—despite all precautions—an accident happens, the system dispatches remedial services to help repair the damage. In this manner, the policy-based fences and after-the-fact ambulances work together to ensure a safer environment for individual agents and the communities in which they operate.

Policy-based agent management

Policy-based management approaches have become popular in the past few years. For example, unlike previous versions, the Java 2 security model defines security policies as distinct from the implementation mechanism. A security manager controls access to resources, relying on a security policy object to dictate whether class *X* has permission to access system resource *Y*. The policies are expressed in a persistent format so any tools that support the policy syntax specification can view and edit them. This approach allows policies to be configurable, relatively more flexible, fine-grained, and extensible. Applications developers no longer need to subclass the security manager and hard-code the application's policies into the subclass. Programs can use the policy file and the extensible permission object to build an application with a security policy that can change without requiring changes in source code.

As with most policy-based management approaches, Java 2 is concerned only with authorization, encryption, and access control. KAoS policy-based agent management includes these features while adding the ability to control Nomads resources. But because of our focus on agent systems, KAoS goes beyond these typical security concerns in significant ways. For example, the KAoS architecture introduced the concept of agent conversation policies.[4,5] The agent-to-agent communication process uses appropriate semantics to form, maintain, and disband teams of agents.[6] Conversation policies assure coherence in the team commitments that heterogeneous agents of different sophistication levels adopt and discharge. These policies also assure robust behavior and minimize computational overhead for team maintenance.[4,7] In addition to conversation policies, we are developing representations and enforcement mechanisms for mobility policies,[8,9] domain registration policies, and various forms of obligation policies.

Our approach differs significantly from the approach of others who are developing security, robustness, and cooperativity constraints for agent communities. First, unlike most multiagent coordination environments, we do not assume that we are dealing with a homogeneous set of agents written within the same agent framework. With respect to environmental protection, legal, and social-services functions, our system requires little or no modification to put various kinds of agents on the same footing. For this reason, policy-management mechanisms can protect against the negative effects of buggy or malicious agents.

Terraforming cyberspace involves more than regulating computing resources and protecting agent state.

A policy management framework must ensure maximum agent freedom.

Second, the framework must support dynamic runtime policy changes, not merely static configurations determined in advance. Third, the framework must be extensible to a variety of execution platforms with different enforcement mechanisms—initially Java and Aroma—but also to any platform for which a guard can be written. Fourth, the framework must be robust in continuing to manage and enforce policies in the face of attack or the failure of any combination of components.

Finally, we recognize the need for easy-to-use policy-based administration tools capable of containing domain knowledge and conceptual abstractions that let applications designers focus their attention more on high-level policy intent than on implementation details. Such tools require powerful graphical user interfaces for monitoring, visualizing, and dynamically modifying policies at runtime.

In short, the policy management framework must ensure maximum freedom and heterogeneity of the agents and the nonintrusiveness of the enforcement mechanisms, while respecting the bounds of human-determined policy constraints designed to ensure selective conformity of behavior.

DAML-based policy representation

In principle, developers could use a variety of languages to express policies. At one extreme, they might write these languages in a propositional or constraint language. At the other extreme lie a wide variety of simpler schemes, each of which gives up some types of expressivity. Several considerations affect the choice of language for a particular application, including composability, computability, efficiency, expressivity, and amenability to detecting equivalence and discovering conflicts.

With funding from the DARPA CoABS program, we have begun developing the implementation-neutral KAoS policy representation (KPR), expressed in DARPA Agent Markup Language (http://www.daml.org). For our current applications, we use a very simple XML representation; however, we have recently drafted an initial KPR specification and DAML encoding, and the implementation is under way.

Designed to support the emerging semantic Web, DAML is the latest in a succession of Web markup languages. HTML, the first Web markup language, allowed users to mark up documents with a fixed set of formatting tags for human use and readability. XML lets users add arbitrary structures to their documents but directly expresses very little about what the structures mean. Resource description format (RDF) encodes meaning in sets of subject-verb-object triples, where a universal resource identifier, typically a URL, can identify the elements of each triple.

DAML extends RDF with new constructs that let users specify ontologies—machine-interpretable representations of terms and their relationships—composed of taxonomies of classes, relationships, and—in the near future—inference rules. These ontologies have a variety of uses, such as enabling more accurate or complex Web searches. Agents can also use semantic markup languages to understand and manipulate Web content in significant ways; to discover, communicate, and cooperate with other agents and services; or, as we describe here, to interact with policy-based management and control mechanisms.

The current KPR specification defines basic ontologies for things such as actors, actions, targets, policies, and policy conditions. It then extends these ontologies to represent simple atomic Java permissions as well as more complex Nomads and KAoS policy constructs. For a given application, developers and users will further extend these ontologies and put instances of policy objects into force as required. Through various property restrictions, an ontology can scope a given policy instance to individual agent instances, agents of a given class, agents running in a given instance of a computational environment such as a VM, or agents that are members of a given domain or subdomain.

The actor ontology distinguishes between agents, which generally can only perform ordinary actions, and domain managers, guards, and authorized human users that it can permit or obligate to perform certain policy actions, such as approval and enforcement. The policy ontology distinguishes between authorizations, which are constraints that permit or forbid some action, and obligations, which are constraints that can either require performing some action or waive such a requirement.[10]

The KPR ontologies serve a variety of purposes. The obvious primary application is during inference relating to policy conflict resolution. Changes or additions to policies in force, or a change in an actor's status—for example, an agent joining a new domain or moving to a new host—require logical inference to determine which policies are in conflict and how to resolve these conflicts. The ontologies are also useful in policy disclosure management, reasoning about future actions based on knowledge of policies in force, and in assisting policy specification tool users in defining new policies consistent with previous ones.

KAoS policy management architecture

Figure 2 shows the major components of the KAoS policy management architecture.

The KAoS policy administration tool (KPAT), a graphical user interface to domain management functionality, makes it easier for administrators to develop policy specifications, revisions, and applications with-

out undergoing extended training. With KPAT, an authorized user can make changes to policies using a secure Web browser. Alternatively, trusted authenticated components such as guards can, if authorized, propose policy changes autonomously based on their observation of system events.

Structuring groups of agents into agent domains and subdomains facilitates policy administration. A given domain can extend across host boundaries, and multiple domains can exist concurrently on the same host. Depending on policy restrictions, agents can become members of more than one domain at a time.

The KAoS domain manager serves as a policy decision point to determine whether agents can join a domain and for policy conflict resolution. The domain manager ensures policy consistency at all levels of a domain hierarchy, notifies guards in the event of a policy change, and stores policies in the repository.

The domain manager stores policies in an implementation-neutral format, currently very simple but soon to be based on our DAML policy representation. We intend to allow authorized entities to access these policies, which are available in a secure library repository such as an LDAP directory, in accordance with policy disclosure strategies.[11] For example, agents may need to understand domain policies in advance of submitting a registration request to a new domain. Because the library expresses the policies declaratively, authorized entities can analyze and verify them in advance and offline, maximizing the efficiency of execution mechanisms.

Guards interpret policies that the domain manager has approved and enforce them with appropriate native mechanisms. While KPAT and the domain manager work identically across different execution environments, we design guards for a specific execution environment. Our approach enables policy uniformity in domains across multiple VMs and hosts as long as they provide semantically equivalent monitoring and enforcement mechanisms to the guards. Under these conditions, these policy-based mechanisms can maintain consistency among agents written in different languages and frameworks and running on different platforms and hosts.

The KPR-based models and tools will take into account differences in available enforcement mechanisms in different computing environments. Because policy analysis and policy conflict resolution normally take place before the system gives the policy to the guard for enforcement, the operation of the guards and enforcement mechanisms can be lightweight and efficient.

Sun's JAAS provides methods that tie access control to authentication. In KAoS, we have developed methods based on JAAS that will allow scoping policies to individual agent instances rather than just to Java

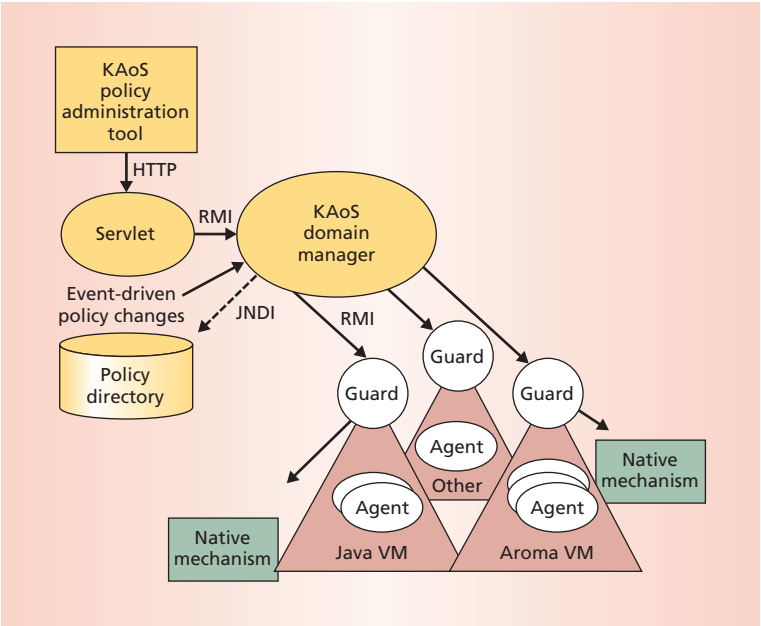


Figure 2. KAoS policy management architecture. The domain manager serves as a policy decision point to determine whether agents can join a domain and for policy conflict resolution. Guards interpret policies that the domain manager has approved and enforce them with appropriate native mechanisms. The KAoS policy administration tool, a graphical user interface to domain management functionality, makes it easier for administrators to develop policy specifications without undergoing extended training.

classes. Currently, JAAS can be used with Java VMs; in the future, it should be possible to use JAAS with the Aroma VM as well. For policies that go beyond simple access-based permissions—such as obligation policies, registration policies, or conversation policies—guards implement additional auxiliary KAoS management and enforcement capabilities as required.

The Aroma VM's resource-control mechanisms allow limits to be placed on both the rate and quantity of resources used by Java threads. Guards running on the Aroma VM can use these mechanisms to provide enhanced security, maintain quality of service for given agents, or give priority to important tasks.

Applications and benefits of a policy-based approach

The DARPA CoABS-sponsored Coalition Operations Experiment (http://www.aiai.ed.ac.uk/project/coax/)[12] provides an example of applying KAoS, Nomads, and Java security policies and mechanisms. CoAX models military coalition operations and implements agent-based systems to mirror coalition structures, policies, and doctrines. The project offers a promising new approach for using the agent-based computing paradigm to deal with issues such as

- the interoperability of new and legacy systems;
- the implicit nature of current coalition policies;
- security in dynamic organizational structures with varying levels of trust and access; and
- prevention and recovery from attack, system failure, or service withdrawal.

The future surely includes much more interesting and amazing agent-powered devices than we can currently imagine.

All these issues have analogs in commercial and industrial settings of the future.

KAoS provides mechanisms for overall management of coalition organizational structures represented as domains and policies, while Nomads provides strong mobility, resource management, and protection from denial-of-service attacks for untrusted agents that run in its environment.

Combining preanalyzed policy libraries, separate policy decision and conflict resolution mechanisms, and efficient policy enforcement methods makes using policy-based administration tools maximally effective. A policy-based approach has the additional advantages of reusability, efficiency, context sensitivity, and verifiability.

Reusability. Policies encode sets of useful constraints on agent or component behavior, packaging them for easy reuse as the occasion requires. Reusing policies when they apply reaps the lessons learned from previous analysis and experience while saving the time it would take to reinvent them.

Efficiency. In addition to lightening the application developers' workload, explicit policies can sometimes increase runtime efficiency. For example, to the extent that it can take place in advance, policy conflict resolution and conversion of policy for use by appropriate enforcement mechanisms can increase overall performance.

Context sensitivity. Explicit policy representation improves the ability of agents, components, and platforms to respond to changing conditions and, if necessary, reason about the implications of the policies that govern their behavior.

Verifiability. Representing policies in an explicitly declarative form instead of burying them in the implementation code provides better support for policy analysis. First—and this is absolutely critical for security policies—we can externally validate that the policies are sufficient for the application's tasks and bring both automated theorem-provers and human expertise to this task. Second, certain methods ensure that program behavior that follows the policy will also satisfy many of the important properties of reactive systems such as liveness, recurrence, safety invariants, and so forth. Finally, with this approach we can predict how explicit policies governing different types of agent behavior can compose.

CYBERFORMING TERRASPACE

Clearly, creating a robust and policy-governed environment where societies of heterogeneous agents can flourish is just a first step. The next step would be to *cyberform terraspace*, giving agents a permanent foothold in the material world.

Tomorrow's world will be filled with agents embedded everywhere in the places and things around us. Providing a pervasive web of sensors and effectors, these agents will function as cognitive prostheses—computational systems that leverage and extend human intellectual, perceptual, and collaborative capacities, just as a steam shovel is a sort of muscular prosthesis or eyeglasses are a sort of visual prosthesis. Thus, the focus of AI research is destined to shift from artificial intelligence to *augmented* intelligence. We anticipate that this augmentation will not simply result in a quantitative boost in human intelligence, but also in a qualitative change in the way we live and work in tandem with artificial agents pervasively embedded in our physical and personal environments.

While simple software or robotic assistants of various kinds capture our attention today, the future surely includes much more interesting and amazing agent-powered devices than we can currently imagine. A key requirement for such devices is real-time cooperation with people and other autonomous systems. Although heterogeneous cooperating entities can operate at different levels of sophistication and with varying degrees of autonomy, they will require some common means of representation and participation in joint tasks. Just as important, developers of such systems will need tools and methodologies to assure that their systems will work together reliably even when designed independently.

A key requirement for designing cooperative autonomous systems is a thorough understanding of the kinds of interactive contexts in which humans and autonomous systems will cooperate. With our colleagues at the Research Institute for Advanced Computer Science (RIACS), we are investigating the use of Brahms[13] as an agent-based design toolkit to model and simulate realistic work situations in space. The agent-based simulation in Brahms will eventually become the basis for the design of functions for space operations. Meanwhile, we are enhancing the KAoS agent framework to incorporate an explicit general model of teamwork and adjustable autonomy appropriate for space operations scenarios.[6]

By using a combination of Brahms and KAoS capabilities, we will be able to incorporate realistic models of astronaut work environments and practices in a theory of human-agent teamwork and adjustable autonomy. This approach seeks to balance previous top-down theoretical agent teamwork formulations with an understanding derived from empirical analysis and simulation. Various team members, from humans to sophisticated autonomous systems to simple devices and sensors, will rely on empirically derived, policy-based capabilities to assure coherence in the adoption, delegation, and discharge of team

commitments and to keep overhead for team maintenance to an absolute minimum.

MAKING CYBERSPACE A BETTER PLACE FOR HUMAN AGENTS

Thus far, our central metaphor has been terraforming cyberspace for the benefit of software agents, but cyberspace itself is increasingly being perceived in spatial terms for human users. We speak and think of visiting Web sites and of getting lost while trying to navigate through the hyperlink tangle, rather than of the information packets being trucked to our computers on the information superhighway. In any case, few people live and work on superhighways. As researchers couple virtual reality technologies with high-bandwidth connectivity, the perception of cyberspace as *space* will become a reality. Cyberspace will be built to fit the human sense of space.

Odessa Street in Seaside, Florida. Courtesy Steven Brooke.

To this end, IHMC is collaborating with the developers of Seaside, Florida, a small, sophisticated Gulf Coast town that is the birthplace of New Urbanism. The New Urbanism movement includes urban designers, environmentalists, transportation experts, social justice advocates, and others who are working together to change American land use from highway-oriented sprawl. The movement proposes reviving and updating traditional town-building principles to produce human-scaled settlements in which numerous pathways connect a variety of buildings internally and to the surrounding landscape. IHMC and the Seaside Institute are hosting a series of workshops, study groups, and visiting scholar programs exploring the application of well-evolved urban design principles to the newly burgeoning and chaotic world of electronic space design.

As a new kind of environment for human beings, cyberspace is now woefully primitive. Most of our electronically built space is a rat's nest of bewildering pathways of indeterminate destination, much like medieval Rome. The humanist Popes of the Renaissance used the *cittá ideale* concepts to produce connectivity and impart legibility to the layout of their city. In the process, they produced some of the world's most memorable, elegant, and comfortable streets and squares, which continue to provide an environment for the lives of a rich variety of people engaged in all manner of activities.

Civilization begins when human beings find places to be, make these places their homes, then create ways to communicate and work together in their chosen locations. Those who are designing and building cyberspace might benefit from lessons learned over thousands of years by those who have been engaged in designing humanity's best physical environments. ✸

Acknowledgments

The authors thank the DARPA CoABS and UltraLog Programs, the NASA Cross-Enterprise and Intelligent Systems Programs, the National Technology Alliance, and the Boeing Company. Thanks also to many other colleagues who have assisted in this work, including Mark Adler, David Allsopp, Patrick Beautement, Maggie Breedy, Todd Carrico, Janet Cerniglia, Bill Clancey, Rob Cranfill, Grzegorz Czajkowski, Chris Dellarocas, Mark Greaves, Jack Hansen, Pat Hayes, Jim Hendler, Greg Hill, Heather Holmback, Wayne Jansen, Martha Kahn, Mike Kerstetter, Mike Kirton, Mark Klein, Gerald Knoll, Henry Lieberman, James Lott, Mike Mahan, Debbie Prescott, Phil Sage, Kent Seamons, Maarten Sierhuis, Austin Tate, Al Underbrink, Doyle Weishar, Tim Wright, Alex Wong, and members of the DARPA CoABS Coalition Operations Experiment, the Java Agent Services Expert Group, and the FIPA Abstract Architecture Working Group.

References

1. I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, Calif., 1999.
2. N. Suri et al., "Strong Mobility and Fine-Grained Resource Control in Nomads," *Proc. 2nd Int'l Symp. Agents Systems and Applications and the 4th Int'l Symp. Mobile Agents* (ASA/MA 2000), Springer-Verlag, Berlin, 2000, pp. 2-15.
3. N. Suri, P.T. Groth, and J.M. Bradshaw, "While You're Away: A System for Load-Balancing and Resource Sharing Based on Mobile Agents," *Proc. 1st IEEE/ACM Int'l Symp. Cluster Computing and the Grid*, IEEE CS Press, Los Alamitos, Calif., pp. 470-473.
4. M. Greaves, H. Holmback, and J.M. Bradshaw, "Agent Conversation Policies," *Handbook of Agent Technology*, J.M. Bradshaw, ed., AAAI Press/The MIT Press, Cambridge, Mass., submitted for publication.
5. J.M. Bradshaw et al., "KAoS: Toward an Industrial-Strength Generic Agent Architecture," *Software Agents*, AAAI Press/The MIT Press, Cambridge, Mass., 1997, pp. 375-418.
6. J.M. Bradshaw et al., "Teamwork and Adjustable Autonomy for the Personal Satellite Assistant," *Proc.*

Int'l Joint Conf. Artificial Intelligence (IJCAI 01) *Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, AAAI Press/The MIT Press, Cambridge, Mass., 2001.
7. J.M. Bradshaw et al., "Agents for the Masses?" *IEEE Intelligent Systems*, Mar./Apr. 1999, pp. 53-63.
8. W.A. Jansen, "A Privilege Management Scheme for Mobile Agent Systems," *Proc. 1st Int'l Workshop Security of Mobile Multiagent Systems and 5th Int'l Conf. Autonomous Agents* (Agents 2001), ACM Press, New York, 2001, pp. 46-53.
9. G. Knoll, N. Suri, and J.M. Bradshaw, "Path-Based Security for Mobile Agents," *Proc. 1st Int'l Workshop Security of Mobile Multiagent Systems and 5th Int'l Conf. Autonomous Agents* (Agents 2001), ACM Press, New York, 2001, pp. 54-60.
10. N. Damianou et al., *Ponder: A Language for Specifying Security and Management Policies for Distributed Systems, Version 2.3,* tech. report TRDOC 2000/I, Dept. of Computing, Imperial College of Science, Technology and Medicine, London, 2000.
11. K.E. Seamons, M. Winslet, and T. Yu, "Limiting the Disclosure of Access Control Policies During Automated Trust Negotiation," *Proc. Symp. Network and Distributed System Security*, 2001.
12. D. Allsopp et al., "Software Agents as Facilitators of Coherent Coalition Operations," *Proc. 6th Int'l Symp. Command and Control Research and Technology*, June 2001.
13. M. Sierhuis et al., "Modeling and Simulating Human Activity," *Fall Symp. Simulating Human Agents*, AAAI, North Falmouth, Mass., 2000, pp. 100-110.

Jeffrey M. Bradshaw *has led agent research and development teams at Boeing, Eurisco, and the Institute for Human and Machine Cognition, University of West Florida. A Fulbright scholar, he currently chairs ACM SIGART and the NASA Ames RIACS Science Council and is a member of the Autonomous Agents Steering Committee. He received a PhD in cognitive science from the University of Washington.*

Niranjan Suri *is a research scientist at the Institute for Human and Machine Cognition. He is the designer of the Aroma virtual machine and the Nomads agent environment. His research interests include distributed systems, computer and network security, and pervasive computing. Suri received an MS in computer science from the University of West Florida.*

Alberto J. Cañas *is the associate director of the Institute for Human and Machine Cognition and an associate professor of computer science at the University of West Florida. He graduated as a systems engineer from the Instituto Tecnológico de Monterrey, Mexico, and received a PhD in management science from the University of Waterloo, Canada.*

Robert Davis *is the founder of Seaside, Florida, the birthplace of New Urbanism, a growing movement in land planning. He is a recipient of the Rome Prize, Florida's Governor's Award, and Coastal Living's Conservation Award for Leadership. Davis received an MBA from Harvard University. Davis is a Fellow of the American Academy in Rome and the Institute of Urban Design and is a member of the Institute for Human and Machine Cognition.*

Kenneth Ford i*s cofounder and director of the Institute for Human and Machine Cognition. His interests include artificial intelligence, cognitive science, Internet-based applications, and entrepreneurship in government and academia. Ford is a Fellow of the American Association for Artificial Intelligence and was awarded the NASA Outstanding Leadership Medal in 1999.*

Robert Hoffman *is a research scientist at the Institute for Human and Machine Cognition. He is a Fulbright Scholar and a Fellow of the American Psychological Society. He specializes in expertise studies including knowledge elicitation and the design of human-centered systems. Hoffman received a PhD in experimental psychology from the University of Cincinnati.*

Renia Jeffers *has been a lead developer for the Knowledgeable Agent-oriented System (KAoS) for several years, including work on agent conversations, domain management, security, mobility, and policy. Jeffers received an MS in software engineering from Seattle University.*

Thomas Reichherzer *is presently with the Institute for Human and Machine Cognition. His research interests include software agents, knowledge representation, reasoning techniques, and natural language processing. Reichherzer received the diploma in informatics from the University of Ulm, Germany, and an MS in computer science from the University of West Florida, Pensacola.*

Contact the authors at {jbradshaw, nsuri, acanas, kford, rhoffman, rjeffers, treichhe}@ai.uwf.edu.