Terraforming Cyberspace

Jeffrey M. Bradshaw, Niranjan Suri, Maggie Breedy, Alberto Cañas, Robert Davis, Kenneth Ford,
Robert Hoffman, Renia Jeffers, Shri Kulkarni, James Lott, Thomas Reichherzer, and Andrzej Uszok
Institute for Human and Machine Cognition (IHMC)
40 South Alcaniz Street
Pensacola, FL 32501
850-202-4400
{jbradshaw, nsuri, mbreedy, acanas, rdavis, kford, rhoffman, rjeffers, skulkarni, jlott, treichhe, auszok}@ai.uwf.edu

INTRODUCTION

During the 1940s, under the pseudonym of Will Stewart, Jack Williamson published a series of fictional stories describing a process for attaching atmospheres to planets in order to make them capable of sustaining life. 'Terraforming,' the term he coined for this activity was first picked up by other science fiction writers. Eventually, it captured the imagination of a small but zealous core of scientists, space advocacy groups, and segments of the public who began focusing on Mars as the most likely target for transformation and eventual colonization. The May 1991 issue of Life Magazine ran a cover story describing a 150-year plan for a Martian metamorphosis through orbiting solar reflectors that would melt polar water, surface factories that would produce needed gases in the atmosphere, and the ultimate planting of hearty plant species as the temperature approached the freezing point of water (figure 1). Today many articles, books, and Web sites continue to develop the theme.

Figure 1. Terraforming Mars.

Like pre-terraformed Mars for humans, cyberspace is currently a lonely, dangerous, and relatively impoverished place for software agents (figure 2). Though promoted as collaborative, agents do not easily sustain rich long-term peer-to-peer relationships, let alone any semblance of meaningful community involvement. While their features for secure reliable interaction are often touted, there is no social safety net to help agents out when they get stuck, or worse yet to prevent them from setting the network on fire when they go off the deep end. Despite the fact that agent designers want them to communicate at an "almost human" level, agents are cut off from most of the world in which humans operate. Though capable of self-directed mobility, they are hobbled by severe practical restrictions on when and where they can go. Ostensibly endowed with autonomy, an agent's very existence can be terminated unceremoniously by the first passerby who happens to find the power switch.

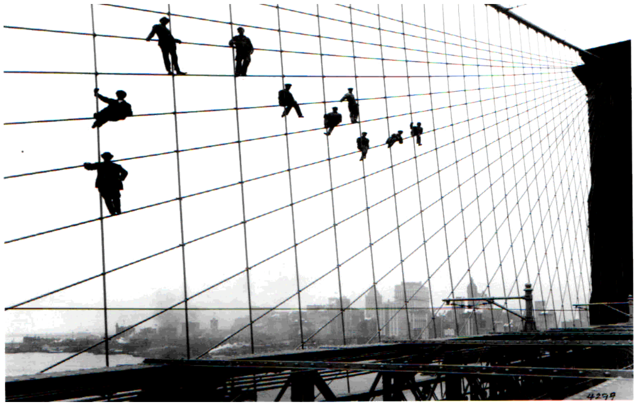


Figure 2. Agent life on the wire is "solitary, poor, nasty, brutish, and short." (Brooklyn Bridge workers, 1914, courtesy Collections of the Municipal Archives of the City of New York)

In 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.[1] With rare exception, today's agents are not deployed in critical, long-lived, secure, or high-risk tasks, or on missions requiring widespread agent migration, or the collaboration of large numbers of agents interacting in complex, unpredictable ways.

Progress in some of these limitations necessarily awaits the results of ongoing research in traditional approaches to agent autonomy,

[1] Thomas Hobbes' entire passage is worth reproducing here: "Of the natural condition of Mankind, as concerning their felicity, and misery, whatsoever therefore is consequent to a time of war, where every man is enemy to every man; the same is consequent to the time, wherein men live without other security, than what their own strength, and their own invention shall furnish them withal. In such condition, there is no place for industry, because the fruit thereof is uncertain; and consequently no culture of the earth, no navigation, nor use of the commodities that may be imported by sea; no commodious building, no instruments of moving and removing such things as require much force; no knowledge of the face of the earth, no account of time, no arts, no letters, no society; and which is worst of all, continual fear and danger of violent death; and the life of people, solitary, poor, nasty, brutish, and short" (Leviathan, i . xiii. 9). Does Hobbes' general argument for the institution of government as a check on the self-serving tendency of individuals find a more natural application to artificial agents than to humans?

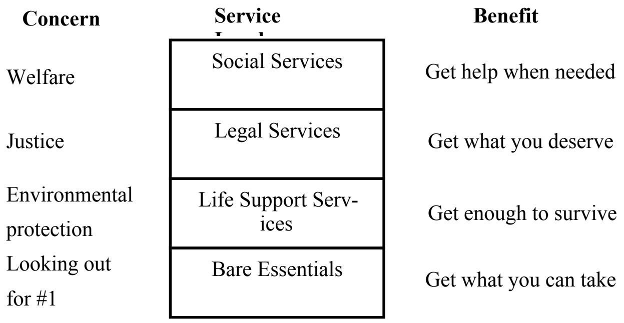
collaborativity, adaptivity, and mobility. However, we argue that focusing greater attention not only on making agents smarter and stronger but also on making the environment in which they operate more capable of sustaining various forms of agent life and civilization would simplify some of these problems.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

Fortunately, the basic infrastructure with which we can begin the terraforming effort is becoming more available. Designed from the ground up to exploit next-generation Internet capabilities, grid-based approaches aim to 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 classes of applications possible [14]. By the time these approaches become mainstream for large-scale applications, they will also have migrated to ad hoc local networks of very small devices [16].

The CoABS Grid, based on Sun's Jini services and developed at Global InfoTek (GITI) under DARPA's Control of Agent-Based Systems (CoABS) program, arguably provides the most successful and widely used infrastructure to date for the large-scale integration of heterogeneous agent frameworks with object-based applications, and legacy systems [9; 21]. Over the next few years, we expect a confluence of this effort with those of the larger computational grid community (http://www.gridforum.org). The Java Agent Services Expert Group (JAS, JSR 87), under the auspices of Sun's Java Community Process (http://www.java-agent.org), the OMG Agent PSIG (http:// www.objs.com /agent/), and the FIPA Abstract Architecture Working Group (http:// www.fipa.org /activities /architecture.html) are similarly at work on essential contributions to interoperable agent infrastructure.

However, we must go far beyond these current efforts to enable the vision of terraforming cyberspace (Figure 3). Current infrastructure implementations typically provide few resource guarantees and no incentives for agents and other components to look beyond their own selfish interests. At a minimum, future infrastructure must go beyond the *bare essentials* to provide pervasive *life support services* (relying on mechanisms such as orthogonal persistence and strong mobility [30; 31]) that help ensure the survival of agents that are 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. Benevolent *social services* will also eventually be provided to offer proactive help when needed. Although some of these elements of terraforming for agents exist in embryo within specific agent systems, their scope and effectiveness has been limited by the lack of underlying support at the platform level.

Here we describe how we are working toward extending current agent infrastructure to provide support for rudimentary initial "terraforming" services. We will first describe NOMADS life support services. Then we will show how we are exploring the basics of legal and social services through the use of KAoS domain and policy management models and mechanisms. Finally, we will sketch our view of complementary principles and capabilitie s that will be necessary to "cyberform terraspace."



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 3. Elements of terraforming for software agents.

NOMADS LIFE SUPPORT SERVICES

NOMADS is the name we have given to the combination of Aroma, an enhanced Java-compatible Virtual Machine (VM), with its Oasis agent execution environment [30; 31]. In its current version, it is designed to provide basic *life support services* ensuring agent environmental protection of two kinds:

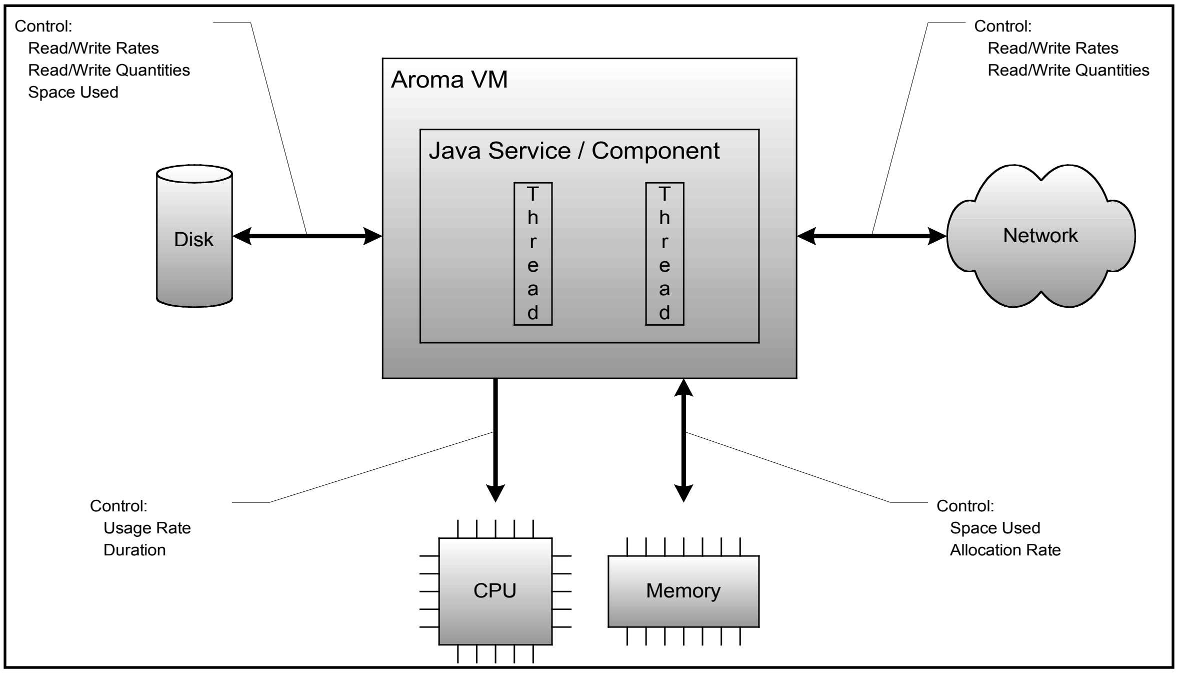
- assurance of availability of system resources, even in the face of buggy agents or denial-of- service attacks;
- protection of agent execution state, even in the face of unanticipated system failure.

Our approach for life support services has thus far been two-pronged: enabling as much protection as possible in standard Java VMs while also providing NOMADS and the enhanced Aroma VM for those agent applications that require it.

For agents running in standard Java VMs, we create software-based Guards, which enforce policies by relying on the capabilities of the Java 2 security model (including permissions and privileged code wrappers) and the Java Authentication and Authorization Service (JAAS). In contrast to other implementations of Java security, our enhanced JAAS-based approach allow revocation of access permissions under many circumstances as well as the granting of different permissions to different instances of agents from the same code base. For policies that go beyond simple access-based permissions (e.g., obligations, registration policies, conversation policies), Guards implement additional auxiliary KAoS management and enforcement capabilities as required.

Unfortunately, some kinds of protection cannot be provided by merely bolting on new services on top of standard Java VMs. Although Java is currently the most popular and arguably the most mobility-minded and security-conscious mainstream language for agent development, current versions fail to address many of the unique challenges posed by agent software. While few if any requirements for Java mobility, security, and resource management are entirely unique to agent software, typical approaches used in non-agent software are usually hard-coded and do not allow the degree of on-demand responsiveness, configurability, extensibility, and fine-grained control required by agent-based systems.

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



Protection of agent resources. To fully appreciate the resource control features of Aroma and NOMADS, some understanding of the current Java security model is needed. Early versions of Java relied on the sandbox model to protect mobile code from accessing dangerous methods. In contrast, the security model in the current Java 2 release is *permission-based*. Unlike the original "all or nothing" approach, Java applets and applications can be given varying amounts of access to system resources. Unfortunately, current Java mechanisms do not address the problem of resource control. For example, while it may be possible to prevent a Java program from writing to any directory except /tmp (an *access* control issue), once the program is given permission to write to the /tmp directory, no further restrictions are placed on the program's I/O (a *resource* control issue). As another example, there is no way in the current Java implementation to limit the amount of disk space the program may use or to control the rate at which the program is allowed to read and write from the network.

Resource control is important for several reasons. First, without resource control, systems and networks are open to denial of service attacks through resource overuse. Second, resource control lays the foundation for quality-of-service guarantees. Before any quality-of-service guarantees can be made about the availability of resources, the system must be able to limit resource utilization of other tasks (which is currently not possible in the Java environment). Third, resource control presupposes resource accounting, which allows the resources consumed by some component of a system (or the overall system) to be measured for either billing or monitoring purposes. Monitoring resource utilization over time allows the detection of abnormal behavior as part of the system.

Finally, the availability of resource control mechanisms in the environment simplifies the task of developing systems for resource-constrained situations. Consider the task of developing and deploying a new system requiring concurrent execution and resource sharing with existing systems. In such scenarios, the developer of the new system often has to limit the resource utilization of the new system in order to not interfere with the operations of the existing systems (for example, maybe the new system can only use 500 Kb/sec of network bandwidth because the rest of the available network bandwidth is required by the existing systems). Providing such a guarantee requires significant effort on behalf of the developer of the new system. However, if the underlying environment were to provide resource control mechanisms, then the new system could simply make a request to the underlying environment, which can then provide the necessary guarantees.

Aroma currently provides a comprehensive set of resource controls for CPU, disk, and network (Figure 4). The resource control mechanisms allow limits to be placed on both the rate and quantity of resources used by Java threads. Rate limits include CPU usage, disk read rate, disk write rate, network read rate and network write rate. Rate limits for I/O are specified in bytes/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. Quantity limits are specified in bytes. One of the major benefits of the Aroma VM is that resource controls are transparent to the Java code executing inside the VM. In particular, the enforcement of the resource limits does not require any modifications to the Java code. Also, the existence of rate limits (and their enforcement) is completely transparent to the Java component or service.

CPU resource control was designed to support two alternative means of expressing the resource limits. The first alternative is to express the limit in terms of bytecodes executed per millisecond. The advantage of expressing a limit in terms of bytecodes per unit time is that given the processing requirements of a thread, the thread's execution time (or time to complete a task) may be predicted. Another advantage of expressing limits in terms of bytecodes per unit time is that the limit is system and architecture independent. The second alternative is to express the limit in terms of some percentage of CPU time, expressed as a number between 0 and 100. Expressing limits as a percentage of overall CPU time on a host provides better control over resource consumption on that particular host.

Rate limits for disk and network are expressed in terms of bytes read or written per millisecond. If a rate limit is in effect, then I/O operations are transparently delayed if necessary until such time that allowing the operation would not exceed the limit. Threads performing I/O operations will not be aware of any resource limits in place unless they choose to query the VM.

Quantity limits for disk and network are expressed in terms of bytes. If a quantity limit is in effect, then the VM throws an exception when a thread requests an I/O operation that would result in the limit being exceeded.

In agent environments, several uses of the NOMADS-based resource control mechanisms are possible. First, the KAoS Domain Manager (explained below) and the VM-level Guard will be able to utilize the resource control capabilities in order to place limits on the resources consumed by services and components running within the Aroma VM. The Guard will be able to vary the resource limits to accommodate changes in policy or level of service guarantees. The Guard will also be able to take advantage of the resource accounting capabilities to measure and report back on the resources consumed by services and components and, if policy permits, to look for patterns of resource abuse that might signal denial-of-service attacks and take autonomous action to reduce resources to the attacker accordingly.

We are working with Sun Microsystems Laboratories to help incorporate resource control capabilities into commercial Java Virtual Machines. Incorporating Aroma-like resource control mechanisms into Java will enable Agent systems and a wide-range of other applications to run in more secure environments.

Protection of agent state. With respect to protection of agent state, we need a way to save the entire state of the running agent or component, including its execution stack, anytime so it can be fully restored in case of system failure or a need to temporarily suspend its execution. The standard term describing this process is checkpointing. Over the last few years, the more general concept of transparent persistence (sometimes called "orthogonal persistence") has also been developed by researchers at Sun Microsystems and elsewhere [20]. The goal of this research is to define language-independent principles and language-specific mechanisms by which persistence can be made available for all data, irrespective of type. Ideally, the approach would not require any special work by the programmer (e.g., implementing serialization methods in Java or using transaction interfaces in conjunction with object databases), and there would be no distinction made between short-lived and long-lived data.

The Aroma VM has been enhanced with the capability to capture the execution state of any running Java program. Current commercial Java VMs do not provide any mechanisms to capture the execution state of a Java program. The state capture mechanism is used to provide strong mobility for NOMADS agents, which allows them to request mobility no matter at what point they are in running their code. Without strong mobility, the code for a mobile agent needs to be structured in a special way to accommodate migration operations. Strong mobility allows agents to be mobile without any special structural requirements (see discussion of resource redirection below).

Promising work has been done on translating agents that use strong mobility into agents that use weak mobility xADDINENRfu. These approaches work well for agents that are single threaded and do not require asynchronous state capture. By asynchronous, we mean a request to capture state that is generated by an external unexpected event or interrupt. The Aroma VM supports capturing of execution state of both multi-threaded agents and allows external events to trigger state capture operations.

We have used the state capture features of NOMADS extensively for agents requiring anytime mobility, whether in the performance of some task or for immediate escape from a host under attack or about to go down (we call this scenario "scram"). We have also put these features to use for transparent load-balancing and forced code migration on demand in distributed computing applications [32]. To support transparent persistence for agents and agent infrastructure components, we are implementing scheduled and on-demand checkpointing services that will protect agent execution state, even in the face of unanticipated system failure.

Forced migration of agents would fail if the agent were using local resources (such as files or network endpoints) on the original host. To solve this problem, we have also implemented transparent redirection of resource access for files and TCP sockets. For example, network redirection is provided through a mechanism called Mockets (mobile sockets) [26], which allow an agent to keep a TCP socket connection open while moving from host to host. Resource redirection is an important requirement to achieve full forced migration of agents.

KAoS LEGAL AND SOCIAL SERVICES

Terraforming cyberspace involves more than regulation of computing resources and protection of agent state. As the scale and sophistication of agents grow, and their lifespan becomes longer, 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 services arenas, it is clear that preventive initiatives are nearly always superior to after-the-fact remedies, as the following verse by Joseph Malins illustrates:

> 'Twas a dangerous cliff, as they freely confessed,
> Though to walk near its crest was so pleasant;
> But over its terrible edge there had slipped
> A duke and full many a peasant.
> So the people said something would have to be done,
> But their project did not at all tally;
> Some said, "Put a fence around the edge of the cliff,"
> Some, "An ambulance down in the valley."

We are basing 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 aim to describe expected behavior in sufficient detail that deviations can be easily anticipated or detected. 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 remedial services are called as a last resort to help repair the damage. In this manner, the policy-based fences and the after-the-fact ambulances work together to ensure a safer environment for individual agents and the communities in which they operate.

Overview of policy-based agent management. Policy-based management approaches have grown considerably in popularity over the last few years. Unlike previous versions, the Java 2 security model defines security policies as distinct from implementation mechanism. Access to resources is controlled by a Security Manager, which relies on a security policy object to dictate whether class X has permission to access system resource Y. The policies themselves are expressed in a persistent format such as text so they can be viewed and edited by any tools that support the policy syntax specification. This approach allows policies to be configurable, and relatively more flexible, fine-grained, and extensible. Developers of applications no longer have to subclass the Security Manager and hard-code the application's policies into the subclass. Programs can make use of the policy file and the extensible permission object to build an application whose security policy can change without requiring changes in source code.

The basic policytool Java currently provides, assists users in editing policy files. However, to be useful and usable in realistic settings, policy-based administration tools should contain domain knowledge and conceptual abstractions to allow applications designers to focus their attention more on high-level policy intent than on the details of implementation. Moreover, while Java provides only for static policies, critical agent applications will require tools for the monitoring, visualization, and dynamic modification of policies at runtime.

The scope of policy-based agent management includes typical security concerns such as *authorization, encryption, access* and *resource control* policies, but also goes beyond these in significant ways. For example, KAoS pioneered the concept of agent conversation policies [5; 17]. Teams of agents can be formed, maintained, and disbanded through the process of agent-to-agent communication using an appropriate semantics [8; 10; 33]. Conversation policies assure coherence in the adoption and discharge of team commitments by heterogeneous agents of different levels of sophistication [5; 6]. These conversation policies are designed to assure robust behavior and to keep computational overhead for team maintenance to an absolute minimum [17; 19; 29]. In addition to conversation policies, we are in the process of developing representations and enforcement mechanisms for *mobility policies [23], domain registration policies,* and various forms of *obligation policies* (see below).

There are some important differences between the objectives of our approach and that of others working to encourage and enforce security, robustness, and cooperativity constraints among communities of agents. First, unlike most multi-agent coordination environments, the approach does not assume that we are dealing with a homogeneous set of agents written within the same agent framework. With respect to the environmental protection, legal, and social services functions provided, we aim insofar as possible to put KAoS and non-KAoS agents on the same footing—with little or no modification to the agents themselves required. In fact, because our services aim to protect against the negative effects of buggy or malicious agents, we have to make sure that the policy-management mechanisms are designed to work even when agents are trying to work against them. Second, insofar as possible the framework needs to support dynamic runtime policy changes, and not merely static configurations determined in advance. Third, the framework needs to be extensible to a variety of execution platforms with different enforcement mechanisms—initially Java and Aroma—but in principle any platform for which a Guard may be written. Fourth, the framework must be robust in continuing to manage and enforce policy in the face of attack or 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 application 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 non-intrusiveness of the enforcement mechanisms, while respecting the bounds of human-determined constraints designed to ensure selective conformity of behavior.

DAML-based policy representation. In principle, developers could use a variety of representations to express policies. At one extreme, they might write these policies in some propositional or constraint representation. 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 representation for a particular application, including composability, computability, efficiency, expressivity, and amenability to various sorts of analysis and inference.

With funding from the DARPA CoABS program, we have developed KAoS Policy Ontologies (KPO). These ontologies are expressed in DAML (http://www.daml.org) and work in conjunction with a set of KAoS policy-management services.

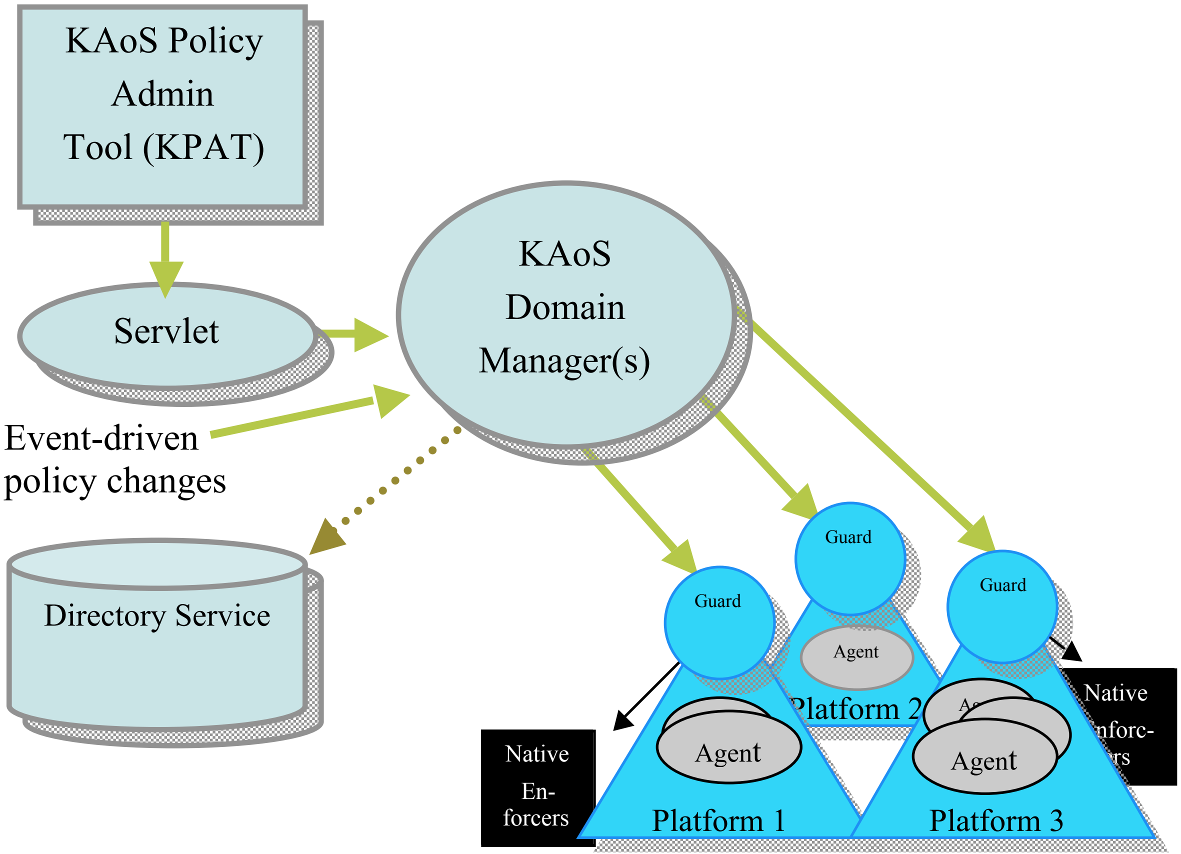
Designed to support the emerging "Semantic Web," DAML is the latest in a succession of Web markup languages [2]. HTML, the first Web markup language, allowed users to markup documents with a fixed set of formatting tags for human use and readability. XML allows users to add arbitrary structures to their documents but expresses very little directly about what the structures mean. RDF (Resource Description Format) encodes meaning in sets of subject-verb-object triples, where elements of these triples may each be identified by a URI (typically a URL).

DAML extends RDF to allow users to specify ontologies composed of taxonomies of classes and inference rules. These ontologies can be used by people for a variety of purposes, 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 outline in this paper, to interact with policy-based management and control mechanisms.

The current KPO specification defines basic ontologies for actors, actions, entities that are the targets of actions (e.g., computing resources), places, policies, and policy conditions. We have extended these ontologies to represent simple atomic Java permissions, as well as more complex NOMADS, and KAoS policy constructs. It is expected that for a given application, the ontologies will be further extended with additional classes, individuals, and rules. Individual policies will be put into force as required. Through various property restrictions, a given policy can be variously scoped, for example, either to individual agents, to agents of a given class, to agents belonging to an intensionally- or extensionally-defined groups (e.g., a domain or team), or to agents running in a given physical place or computational environment (e.g., VM).

The actor ontology distinguishes between agents (that generally can only perform *ordinary actions*) and Domain Managers, Guards, and authorized human users, who may variously be permitted or obligated to perform certain *policy actions,* such as approval and enforcement. The policy ontology distinguishes between *authorizations* (i.e., constraints that permit or forbid some action) and *obligations* (i.e., constraints that require some action to be performed, or else serve to waive such a requirement) [12].

The KAoS Policy Ontologies are intended for a variety of purposes. One obvious application is during inference relating to various forms of online or offline analysis. For example, changes



or additions to policies in force, or a change in status of an actor (e.g., an agent joining a new domain or moving to a new host) require logical inference to determine first of all which policies are in conflict and second how to resolve these conflicts [24]. We have implemented a general-purpose algorithm for policy conflict detection and harmonization whose current results promise a surprising degree of efficiency and scalability.[2] The ontologies may also be used in policy disclosure management (see below), reasoning about future actions based on knowledge of policies in force, and in assisting users of policy specification tools to understand the implications of defining new policies given the current context and the set of policies already in force.

KAoS policy management architecture. Figure 5 shows the major components of the KAoS policy management architecture.

The KAoS Policy Administration Tool (KPAT), a graphical user interface to policy management functionality, has been developed to make policy specification, revision, and application easier for administrators without specialized training. Using KPAT, an authorized user may make changes over the Web to agent policy. Alternatively, trusted components such as Guards may, if authorized, propose policy changes autonomously based on their observation of system events.

Groups of agents are structured into agent domains and subdomains to facilitate policy administration. A given domain can extend across host boundaries and, conversely, multiple domains can exist concurrently on the same host. Depending on whether policy allows, agents may become members of more than one domain at a time.

KAoS Domain Managers (DM) act in the role of policy decision points to determine whether agents can join their domain and for policy conflict resolution.[3] The DM is responsible for ensuring policy consistency at all levels of a domain hierarchy, for notifying Guards about changes in policy or other aspects of system state that may affect their operation, and for storing state in the directory service. Because DM's are stateless, one DM instance may serve multiple domains or conversely, a single large domain may require several instances of the DM to achieve scalable performance.

Policies are stored within ontologies in the directory service (DS). Although DM's normally provide the limited public interface to the DS, private interfaces may allow the DS to be accessed by other authorized entities in accordance with policy disclosure strategies [27]. For example, trusted agents may be allowed to perform queries concerning domain policies in advance of submitting a registration request to a new domain. Because the policies in the directory service are expressed declaratively, some forms of analysis and verification can be performed in advance and offline, permitting execution mechanisms to be as efficient as possible.

Guards interpret policies that have been approved by the DM and enforce them with appropriate native enforcement mechanisms. While KPAT and the DM, and the Guards are intended to work identically across different agent platforms (e.g., DARPA CoABS Grid, Cougaar, Objectspace Voyager) and execution environments (e.g., Java VM, Aroma VM), enforcement mechanisms are necessarily designed for a specific platform and execution environment. Our approach enables policy uniformity in domains that might be simultaneously distributed across multiple platforms and execu-

[2] A detailed description our policy conflict detection and resolution process is currently being prepared for publication.

[3] In the current implementation, DM's delegate inference about policy decisions to the directory service, which incorporates a DAML-based inference engine.

tion environments, as long as semantically equivalent monitoring and enforcement mechanisms are available. Under these conditions, it follows that behavior of agents written using different platforms and running in different execution environments can be kept consistent through the use of these policy-based mechanisms. Because policy analysis and policy conflict resolution normally take place prior to the policy being given to the Guard for enforcement, the operation of the Guards and enforcement mechanisms can be lightweight and efficient.

In applications to date, we have relied on several different kinds of enforcement mechanisms. Enforcement mechanisms built into the execution environment (e.g., OS or Virtual Machine level protection) are the most powerful sort, as they can generally be used to assure policy compliance for any agent or program running in that environment, regardless of how that agent or program was written. For example, the Java Authentication and Authorization Service (JAAS) provides methods that ties access control to authentication. In KAoS, we have developed methods based on JAAS that will allow policies to be scoped to individual agent instances rather than just to Java 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. As described above, the Aroma VM provides, in addition to Java VM protections, a comprehensive set of resource controls for CPU, disk and network. The resource control mechanisms allow limits to be placed on both the rate and the quantity of resources used by Java threads. Guards running on the Aroma VM can use the resource control mechanisms to provide enhanced security (e.g., prevent or disable denial-of-service attacks), maintain quality of service for given agents, or give priority to important tasks.

A second kind of enforcement mechanism takes the form of extensions to particular agent platform capabilities. Agents that participate in that platform are generally given more permissions to the degree they are able to make small adaptations in their agents to comply with policy requirements. For example, in applications using the DARPA CoABS Grid, we have defined a KAoSAgentRegistrationHelper to replace the default GridAgentRegistrationHelper. Grid agent developers need only replace the class reference in their code to participate in agent domains and be transparently and reliably governed by policies currently in force. On the other hand, agents that use the default GridAgentRegistrationHelper do not participate in domains and as a result they are typically granted very limited permissions in their interactions with domain-enabled agents.

Finally, a third type of enforcement mechanism is necessary for obligation policies. Because obligations cannot be enforced through preventive mechanisms, enforcers can only monitor agent behavior and determine after-the-fact whether a policy has been followed. For example, if an agent is required by policy to report its status every 5 minutes, an enforcer might be deployed to watch whether this is in fact happens, and if not to either try to diagnose and fix the problem, or alternatively take appropriate sanctions against the agent (e.g., reduce permissions or publish the observed instance of noncompliance to an agent reputation service).

Applications and benefits of policy-based approach. An example of the application of KAoS, NOMADS, and Java security policies and mechanisms can be found in our work on the DARPA CoABS-sponsored Coalition Operations Experiment (CoAX) (http://www.aiai.ed.ac.uk /project/ coax/) [1]. CoAX models military coalition operations and implement agent-based systems to mirror coalition structures, policies, and doctrines. The project aims to show that the agent-based computing paradigm offers a promising new approach to dealing with issues such as

_ the interoperability of new and legacy systems,

_ the implicit nature of coalition policies,

_ security, and

_ recovery from attack, system failure, or service withdrawal.

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.

The combination of the use of libraries of pre-analyzed policy sets, separate policy decision and conflict resolution mechanisms, and efficient policy enforcement mechanisms make the use of policy-based administration tools maximally effective and performant. 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 in a form where they can be easily reused as the occasion requires. By reusing policies when they apply, we reap the lessons learned from previous analysis and experience while saving ourselves the time it would have taken to reinvent them from scratch.

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

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

Verifiability. By representing policies in an explicitly declarative form instead of burying them in the implementation code, we can better support important types of 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 we can bring both automated theorem-provers and human expertise to this task. Second, there are methods to ensure that program behavior which follows the policy will also satisfy many of the important properties of reactive systems: liveness, recurrence, safety invariants, and so forth. Finally, with explicit policies governing different types of agent behavior, we can predict how policies may compose.

CYBERFORMING TERRASPACE

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, teams of such agents will function as cognitive prostheses—computational systems that leverage and extend human intellectual, perceptual, and collaborative capacities, just as the steam shovel was a sort of muscular prosthesis or the eyeglass a sort of visual prosthesis [13]. Thus the focus of AI research is destined to shift from Artificial Intelligence to Augmented Intelligence [4; 18]. It is clear that terraforming cyberspace is just the first step; the next step will be to cyberform terraspace, giving agents permanent footholds in the material world.

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

One example that presages—albeit primitively—such developments is the Personal Satellite Assistant (PSA), a softball-sized flying robot that is being designed to operate onboard spacecraft in pressurized micro-gravity environments (figure 6) [15]. First proposed and championed by Yuri Gawdiak of NASA Ames, the PSA will incorporate environmental sensors for gas, temperature, and fire detection, providing the ability for the PSA to monitor spacecraft, payload and crew conditions. Video and audio interfaces and speech understanding capabilities will support for navigation, remote monitoring, and video-conferencing. Ducted fans will provide propulsion and batteries will provide portable power. Most importantly, it is intended to work in close interaction with groups of people and artificial agents, which poses daunting challenges to researchers and developers.

With funding and collaboration from NASA Ames, and RIACS, we are investigating issues in human-robotic teamwork and adjustable autonomy for the PSA [7; 8]. Although we currently envision the PSA as the most accessible and practical initial testbed for our prototyping work in the design of collaborative robots, we are confident that our results will generalize to future cooperative autonomous systems of many other sorts. For instance, future human missions to the Moon and to Mars will undoubtedly need the increased capabilities for human-robot collaborations we envision. Astronauts will live, work, and perform laboratory experiments in collaboration with robots not only inside, but also outside the habitat on planetary surfaces.

Our approach to design of cooperative autonomous systems requires first and foremost a thorough understanding of the kinds of interactive contexts in which humans and autonomous systems will cooperate. With our colleagues at RIACS, we have begun to investigate the use of Brahms [28] 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 PSA functions for actual operations. On its part, IHMC is enhancing the KAoS and NOMADS agent frameworks to incorporate explicit general models of teamwork, mobility, and resource control appropriate for space operations scenarios. We expect these models to be mostly represented in the form of policies [7].

The power of general-purpose teamwork models in multi-agent systems usually comes at a high price. Joint intention theory, for example, is built on an extremely powerful logical framework that includes explicit representation of mental attitudes like belief, goal, intention, and so forth [10; 11; 33]. These attitudes are modeled in the traditional way: as new modal operators in a quantified modal logic. Hence, while the most general form of joint intention theory is representationally very attractive, it is often computationally intractable. This tension between expressivity and computability is not limited to teamwork theories; in fact, it is a hallmark of all mentalistic theories of agent behavior and speech-act based agent communication. Thus, when designing agents that include strong teamwork assumptions and powerful communication languages (as do the PSA and other robots), it is critically important to reduce the power of these general models in a way that is sensitive to the agent's domain and expected range of action.

Our approach to building a model of teamwork seeks to incorporate the best of previous research on human-centered collaboration and teamwork, while simultaneously grounding such research in our own work practice study experience. In addition, we are assessing the contributions of allied fields ranging from cognitive function analysis [3], to studies of animal displays, the roles of values and affect, the enablers of effective delegation in humans [25].

By using the analysis and simulation capability in Brahms, we will be able to incorporate models of the PSA work environment and practices in our decisions about how to strategically weaken general joint intention theory without compromising the PSA's ability to perform in its environment. In this way, we will balance empirical analysis, simulation, and top-down theoretical considerations in arriving at a teamwork theory that will allow the PSA to meet the scenario goals. Teams of KAoS-NOMADS-Brahms agents will be formed, maintained, and disbanded through the process of agent-to-agent communication using an appropriate semantics. Agents representing various team members, from humans to autonomous systems to simple devices and sensors, will assure coherence in the adoption and discharge of team commitments and will encapsulate state information associated with each entity. Agent conversation policies will be designed to assure robust behavior and to keep computational overhead for team maintenance to an absolute minimum [6; 17; 19].

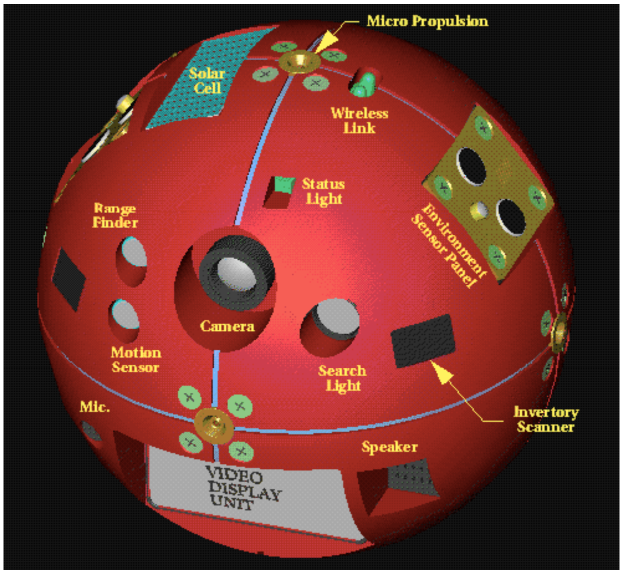


Figure 6. NASA's Personal Satellite Assistant or PSA (Courtesy NASA Ames).

TERRAFORMING TERRASPACE

In discussing requirements for habitable agent environments, we cannot forget that humans have corresponding needs. Terraspace, in many respects, is as badly in need of (re)terraforming as cyberspace.[4] To this end, IHMC is collaborating with the developers of

[4] In an interesting twist, Jack Williamson, who invented the term "terraforming" to describe an alien world altered for human

Seaside, Florida, a small, sophisticated Gulf Coast town that is the birthplace of New Urbanism [22]. 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.

Figure 7. Odessa Street in Seaside, Florida. Courtesy Steven Brooke.

Increasingly, cyberspace is being perceived in spatial terms for human users. We speak and think of *visiting* websites, for example, and of the problems of *getting lost* while trying to *navigate* through the tangle of hyperlinks, rather than of the packets of information being trucked to our computers on the information superhighway. In any case, very few people live and work on superhighways. As virtual reality technologies are coupled 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.

As a new kind of space for human beings, cyberspace is now woefully primitive. Most of our electronic 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 ideas of the *citta ideale* 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 all kinds of people engaged in all manner of activities.

Civilization begins when human beings find places to *be*, make these places their homes, and then create ways to communicate and work together, which depend on 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. Moreover, urban designers may yet have something to learn from the architects of cyberspace.

Ultimately both humans and agents have much to gain through the terraforming and cyberforming of the dual realms of atoms and bits. People will feel more welcome and at home in both places. Agents will be freed from their current role as short-lived visitors on the wire to permanent colonists in real and virtual worlds where we can feel comfortable with not knowing or caring exactly where their programs are being physically hosted. They will truly live among us and we will wonder how we ever lived without them.

habitation in his 1942 novel "Seetee Ship," has used the seemingly paradoxical title "Terraforming Earth" as the name of his latest work of science fiction [34]. The book describes the efforts of scientists to bring life back to the earth after asteroids trigger a new Ice Age.

ACKNOWLEDGEMENTS

This is an updated and expanded version of an article that originally appeared in IEEE Intelligent Systems, July 2001, pp. 49-56. The authors gratefully acknowledge the support of the DARPA CoABS, Augmented Cognition, and Ultra*Log Programs, the NASA Cross-Enterprise and Intelligent Systems Programs, and the National Technology Alliance, Fujitsu Labs, and The Boeing Company while preparing this paper. Thanks also to many other colleagues who have assisted in this work, including Mark Adler, David Allsopp, Alessandro Acquisti, Patrick Beautement, Todd Carrico, Janet Cerniglia, Bill Clancey, Rob Cranfill, Grzegorz Czajkowski, Chris Dellarocas, Greg Dorais, Mark Greaves, Jack Hansen, Pat Hayes, Jim Hendler, Greg Hill, Heather Holmback, Wayne Jansen, Martha Kahn, Mike Kerstetter, Mike Kirton, Mark Klein, Gerald Knoll, Ora Lassila, Henry Lieberman, Mike Mahan, Framk McCabe, Nicola Muscettola, Debbie Prescott, Anil Raj, Phil Sage, Dylan Schmorrow, Kent Seamons, Maarten Sierhuis, Austin Tate, Al Underbrink, Ron Van Hoof, Doyle Weishar, Alex Wong, Tim Wright, and members of the DARPA CoABS Coalition Operations Experiment (CoAX), the Java Agent Services Expert Group (JSR 87) and the FIPA Abstract Architecture Working Group.

REFERENCES

[1] Allsopp, D., Beautement, P., Bradshaw, J. M., Durfee, E., Kirton, M., Knoblock, C., Suri, N., Tate, A., & Thompson, C. (2002). Coalition Agents eXperiement (CoAX): Multi-agent cooperation in an international coalition setting. *A. Tate, J. Bradshaw, and M. Pechoucek (Eds.), Special issue of IEEE Intelligent Systems*.

[2] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American*.

[3] Boy, G. (1998). *Cognitive Function Analysis*., Stamford, CT: Ablex Publishing.

[4] Bradshaw, J. M., Beautement, P., Kulkarni, S., Suri, N., & Raj, A. (2002). *Toward a deliberative and reactive agent architecture for augmented cognition. DARPA Augmented Cognition Program White Paper.*, Pensacola, FL: Institute for Human and Machine Cognition, University of West Florida.

[5] Bradshaw, J. M., Dutfield, S., Benoit, P., & Woolley, J. D. (1997). KAoS: Toward an industrial-strength generic agent architecture. In J. M. Bradshaw (Ed.), *Software Agents*. (pp. 375-418). Cambridge, MA: AAAI Press/The MIT Press.

[6] Bradshaw, J. M., Greaves, M., Holmback, H., Jansen, W., Karygiannis, T., Silverman, B., Suri, N., & Wong, A. (1999). Agents for the masses: Is it possible to make development of sophisticated agents simple enough to be practical? *IEEE Intelligent Systems*(March-April), 53-63.

[7] Bradshaw, J. M., Sierhuis, M., Feltovich, P., Acquisti, A., Jeffers, R., Prescott, D., Suri, N., Uszok, A., & Van Hoof, R. (2002). What we can learn about human-agent teamwork from practice. *Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002).* Bologna, Italy,, Proceedings of the AAMAS Workshop on Teamwork and Coalition Formation,

[8] Bradshaw, J. M., Sierhuis, M., Gawdiak, Y., Jeffers, R., Suri, N., & Greaves, M. (2001). Adjustable autonomy and teamwork for the Personal Satellite Assistant. *Proceedings of the IJCAI-01 Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*. Seattle, WA, USA,,,

[9] Bradshaw, J. M., Suri, N., Kahn, M., Sage, P., Weishar, D., & Jeffers, R. (2001). Terraforming cyberspace: Toward a policy-based grid infrastructure for secure, scalable, and robust execution of Java-based multi-agent systems. *Proceedings of the Second International Workshop on Infrastructure for Agents, Mobile Agent Systems, and Scalable Mobile Agents Systems at the Fifth International Conference on Autonomous Agents (Agents-2001).* Montreal, Quebec, Canada,,,

[10] Cohen, P. R., & Levesque, H. J. (1991). *Teamwork*. Technote 504. Menlo Park, CA: SRI International, March.

[11] Cohen, P. R., Levesque, H. R., & Smith, I. (1997). On team formation. In J. Hintikka & R. Tuomela (Ed.), *Contemporary Action Theory*. Synthese.

[12] Damianou, N., Dulay, N., Lupu, E. C., & Sloman, M. S. (2000). *Ponder: A Language for Specifying Security and Management Policies for Distributed Systems, Version 2.3.* Imperial College of Science, Technology and Medicine, Department of Computing, 20 October 2000.

[13] Ford, K. M., Glymour, C., & Hayes, P. (1997). Cognitive prostheses. *AI Magazine*, 18(3), 104.

[14] Foster, I., & Kesselman, C. (Ed.). (1999). *The Grid: Blueprint for a New Computing Infrastructure.* San Francisco, CA: Morgan Kaufmann.

[15] Gawdiak, Y., Bradshaw, J. M., Williams, B., & Thomas, H. (2000). R2D2 in a softball: The Personal Satellite Assistant. H. Lieberman (Ed.), *Proceedings of the ACM Conference on Intelligent User Interfaces (IUI 2000),* (pp. 125-128). New Orleans, LA,, New York: ACM Press,

[16] Gershenfeld, N. A. (1999). *When Things Start to Think.*, New York: Henry Holt and Company.

[17] Greaves, M., Holmback, H., & Bradshaw, J. M. (2001). Agent conversation policies. In J. M. Bradshaw (Ed.), *Handbook of Agent Technology.* (pp. in preparation). Cambridge, MA: AAAI Press/The MIT Press.

[18] Hamilton, S. (2001). Thinking outside the box at IHMC. *IEEE Computer*, 61-71.

[19] Holmback, H., Greaves, M., & Bradshaw, J. M. (1999). A pragmatic principle for agent communication. J. M. Bradshaw, O. Etzioni, & J. Mueller (Ed.), *Proceedings of Autonomous Agents '99,* (pp. 368-369). Seattle, WA,, New York: ACM Press,

[20] Jordan, M., & Atkinson, M. (1998). *Orthogonal persistence for Java—A mid-term report*. Sun Microsystems Laboratories,

[21] Kahn, M., & Sage, P. (2000). DARPA Control of Agent-Based Systems Grid Tutorial. J. M. Bradshaw (Ed.), *PAAM 2000*. Manchester, England,,,

[22] Katz, P., & Scully, V., Jr. (1993). *The New Urbanism: Toward an Architecture of Community*., New York, NY: McGraw-Hill.

[23] Knoll, G., Suri, N., & Bradshaw, J. M. (2001). Path-based security for mobile agents. *Proceedings of the First International Workshop onthe Security of Mobile Multi-Agent Systems (SEMAS-2001) at the Fifth International Conference on Autonomous Agents (Agents 2001),* (pp. 54-60). Montreal, CA,, New York: ACM Press,

[24] Lupu, E. C., & Sloman, M. S. (1999). Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering—Special Issue on Inconsistency Management.*

[25] Milewski, A. E., & Lewis, S. H. (1994). *Design of intelligent agent user interfaces: Delegation issues*. AT&T Corporate Information Technologies Services Advanced Technology Planning, October 20.

[26] Mitrovich, T. R., Ford, K. M., & Suri, N. (2001). Transparent redirection of network sockets. *OOPSLA WorkshBp on Experiences with Autonomous Mobile Objects and Agent-based Systems*.,,,

[27] Seamons, K. E., Winslet, M., & Yu, T. (2001). Limiting the disclosure of access control policies during automated trust negotiation. *Proceedings of the Network and Distributed Systems Symposium*.,,,

[28] Sierhuis, M. (2001) *Brahms: A Multi-Agent Modeling and Simulation Language for Work System Analysis and Design.* Doctoral, University of Amsterdam.

[29] Smith, I. A., Cohen, P. R., Bradshaw, J. M., Greaves, M., & Holmback, H. (1998). Designing conversation policies using joint intention theory. *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98),* (pp. 269-276). Paris, France,, Los Alamitos, CA: IEEE Computer Society,

[30] Suri, N., Bradshaw, J. M., Breedy, M. R., Groth, P. T., Hill, G. A., & Jeffers, R. (2000). Strong Mobility and Fine-Grained Resource Control in NOMADS. *Proceedings of the 2nd International Symposium on Agents Systems and Applications and the 4th International Symposium on Mobile Agents (ASA/MA 2000).* Zurich, Switzerland,, Berlin: Springer-Verlag,

[31] Suri, N., Bradshaw, J. M., Breedy, M. R., Groth, P. T., Hill, G. A., Jeffers, R., Mitrovich, T. R., Pouliot, B. R., & Smith, D. S. (2000). NOMADS: Toward an environment for strong and safe agent mobility. *Proceedings of Autonomous Agents 2000.* Barcelona, Spain,, New York: ACM Press,

[32] Suri, N., Groth, P. T., & Bradshaw, J. M. (2001). While You're Away: A system for load-balancing and resource sharing based on mobile agents. R. Buyya, G. Mohay, & P. Roe (Ed.), *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid,* (pp. 470-473). Brisbane, Australia,, Los Alamitos, CA: IEEE Computer Society,

[33] Tambe, M., Shen, W., Mataric, M., Pynadath, D. V., Goldberg, D., Modi, P. J., Qiu, Z., & Salemi, B. (1999). Teamwork in cyberspace: Using TEAMCORE to make agents team-ready. *Proceedings of the AAAI Spring Symposium on Agents in Cyberspace*. Menlo Park, CA,, Menlo Park, CA: The AAAI Press,

[34] Williamson, J. (2001). *Terraforming Earth*., New York, NY: Tor Books.