Representation and Reasoning for DAML-Based Policy and Domain Services in KAoS and Nomads

J. Bradshaw[3], A. Uszok[3], R. Jeffers[3], N. Suri[3], P. Hayes[3], M. Burstein[2], A. Acquisti[4], B. Benyo[2], M. Breedy[3], M. Carvalho[3], D. Diller[2], M. Johnson[3], S. Kulkarni[3], J. Lott[3], M. Sierhuis[1], and R. Van Hoof[1]

[1] RIACS and QSS, NASA Ames, MS T35B-1, Moffett Field, CA 94035, {msierhuis, rvanhoof}@mail.arc.nasa.gov

[2] BBN Technologies, 10 Moulton St., Cambridge, MA 02138, {burstein, bbenyo, ddiller}@bbn.com

[3] IHMC/UWF, 40 S. Alcaniz, Pensacola, FL 32501 {jbradshaw, auszok, rjeffers, nsuri, phayes, mbreedy, mcarvalho, mjohnson, skulkarni, jlott}@ai.uwf.edu

[4] SIMS, UC Berkeley, 102 South Hall, Berkeley, CA 94720, acquisti@mail.arc.nasa.gov

ABSTRACT

To increase the assurance with which agents can be deployed in operational settings, we have been developing the KAoS policy and domain services. In conjunction with Nomads strong mobility and safe execution features, KAoS services and tools allow for the specification, management, conflict resolution, and enforcement of DAML-based policies within the specific contexts established by complex organizational structures. In this paper, we will discuss results, issues, and lessons learned in the development of these representations, tools, and services and their use in military and space applications

Keywords

social order, conventions, norms, social control; cultural norms and institutions, ontologies for agents and social modeling; ontologies in agent-based information systems and knowledge management, DAML, policy, domains, KAoS, Nomads, human-agent teamwork, adjustable autonomy, coalition, augmented cognition, cognitive prosthesis

1. INTRODUCTION

The increased intelligence afforded by software agents is both a boon and a danger. By their ability to operate independently without constant human supervision, they can perform tasks that would be impractical or impossible using traditional software applications. On the other hand, this additional autonomy, if unchecked, also has the potential of effecting severe damage in the case of buggy or malicious agents. Techniques and tools must be developed to assure that agents will always operate within the bounds of established behavioral constraints and will be continually responsive to human control. Moreover, the policies that regulate the behavior of agents should be continually adjusted so as to maximize their effectiveness in both human and computational environments.

Under DARPA and NASA sponsorship, we have been developing the KAoS policy and domain services to increase the assurance with which agents can be deployed in a wide variety of operational settings. In conjunction with Nomads strong mobility and safe execution features, KAoS services and tools allow for the specification, management, conflict resolution, and enforcement of policies within the specific contexts established by complex organizational structures. Following a description of these capabilities (section 2), we will conclude with a brief summary of current applications (section 3) and a brief outline of future directions (section 4).

2. KAoS AND NOMADS POLICY AND DOMAIN SERVICES

KAoS is a collection of componentized agent services compatible with several popular agent frameworks, including Nomads [27], the DARPA CoABS Grid [18], the DARPA ALP/Ultra*Log Cougaar framework (http://www.cougaar.net), CORBA (http://www.omg.org), and Voyager (http://www.recursionsw.com/osi.asp). The adaptability of KAoS is due in large part to its pluggable infrastructure based on Sun's Java Agent Services (JAS) (http://java.agent.org). While initially oriented to the dynamic and complex requirements of software agent applications, KAoS services are also being adapted to general-purpose grid computing (http://www.gridforum.org) and Web services (http://www.w3.org/2002/ws/) environments as well [17]. For a full description of KAoS, the reader is referred to [5; 6; 7; 8; 9].

Nomads combines the capabilities of Aroma, an enhanced Java-compatible Virtual Machine (VM), with the Oasis agent execution environment [26]. It is designed to provide environmental protection of two kinds:

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

These basic capabilities of Nomads provide essential features of reliability and safety required for interaction with humans in dynamic and demanding application environments. We are currently working with Sun Microsystems on incorporating resource management features similar to Nomads into a future version of the commercial Java platform.

Following a discussion of the background and motivation for KAoS and Nomads policy and domain services (section 2.1), we will provide an overview of the *KAoS Policy Ontologies* (KPO), which represent both policies and relevant application and organizational state declaratively using the DARPA Agent Markup Language (DAML) (section 2.2). We introduce the KAoS Policy Administration Tool (KPAT)[1], which provides a graphical user interface to create, structure, and administer domains and policies without needing to master all the details of DAML (section 2.3). KAoS and Nomads policy and domain services are used to define, manage, and enforce constraints assuring coherent, safe, effective, and natural interaction among collaborating groups of human and agents. Subsequent sections describe algorithms and mechanisms for policy conflict resolution (2.4), policy distribution (2.5), and policy enforcement (2.6), followed by an example (2.7).

2.1 Background and Motivation

The idea of building strong social laws into intelligent systems can be traced at least as far back as the 1940s to the science fiction writings of Isaac Asimov [3]. In his well-known stories of the succeeding decades he formulated a set of basic laws that were built deeply into the positronic-brain circuitry

[1] Pronounced "KAY-pat.„

of each robot so that it was physically prevented from transgression. Though the laws were simple and few, the stories attempted to demonstrate just how difficult they were to apply in various real-world situations.[2]

Shoham and Tennenholtz [24] introduced the theme of social laws into the agent research community, where investigations have continued under two main headings: *norms* and *policies*. Drawing on precedents in legal theory, social psychology, social philosophy, sociology, and decision theory [34], *norm-based* approaches have grown in popularity [4; 12; 19; 20]. In the multi-agent system research community, Conte and Castelfranchi [11] found that norms were variously described as constraints on behavior, ends or goals, or obligations. For the most part, implementations of norms in multi-agent systems share three basic features:

- they are designed offline; or
- they are learned, adopted, and refined through the purposeful deliberation of each agent; and
- they are enforced by means of incentives and sanctions.

Interest in *policy-based* approaches to multi-agent and distributed systems has also grown considerably in recent years (http://www.policy-workshop.org). While sharing much in common with norm-based approaches, policy-based perspectives differ in subtle ways. Whereas in everyday English the term *norm* denotes a practice, procedure, or custom regarded as typical or widespread, a *policy* is defined by the American Heritage Online dictionary as a "course of action, guiding principle, or procedure considered expedient, prudent, or advantageous.„ Thus, in contrast to the relatively descriptive basis and self-chosen adoption (or rejection) of norms, policies tend to be seen as prescriptive and externally imposed entities. Whereas norms in everyday life emerge gradually from group conventions and recurrent patterns of interaction, policies are consciously designed and put into and out of force at arbitrary times by virtue of explicitly-recognized authority.[3] These differences are generally reflected in the way most policy-based approaches differ from norm-based ones with respect to the three features mentioned above. Policy-based approaches:

- support dynamic runtime policy changes, and not merely static configurations determined in advance;
- work involuntarily with respect to the agents, that is, without requiring the agents to consent or even be aware of the policies being enforced; thus aiming to guarantee that even the simplest agents can comply with policy; and
- wherever possible they are enforced preemptively, preventing buggy or malicious agents from doing harm in advance rather than rewarding them or imposing sanctions on them after the fact.

To increase the likelihood of human acceptability of agent technology, successful systems must attend to both the technical and social aspects of policy [22]. From a technical perspective, we want to be able to help ensure the protection of agent state, the viability of agent communities, and the reliability of the resources on which they depend [9]. To accomplish this, we must guarantee, insofar as is possible, that the autonomy of agents can always be bounded by explicit enforceable policy that can be continually adjusted to maximize the agents' effectiveness and safety in both human and computational environments. From a social perspective, we want agents to be designed to fit well with how people actually work together. Explicit policies governing human-agent interaction, based on careful observation of work practice and an understanding of current social science research, can help assure that effective and natural coordination, appropriate levels and modalities of feedback, and adequate predictability and responsiveness to human control are maintained [8; 14]. These and similar technical and social factors are key to providing the reassurance and trust that are the prerequisites to the widespread acceptance of agent technology for non-trivial applications.

Some important features of KAoS are worth noting here before giving a detailed description. First, the approach does not assume that the policy-governed system is comprised of a homogeneous set of components that have been designed in advance to work with KAoS services. Rather the goal is to be able to have KAoS services work with arbitrarily written components after the fact through support being added transparently at the platform level. Second, insofar as possible the KAoS framework supports dynamic runtime policy changes, and not merely static configurations determined in advance. Third, the framework is extensible to a variety of execution platforms that might be simultaneously running with different enforcement mechanisms—in principle any platform for which policy enforcement mechanisms may be written. Fourth, the KAoS framework is intended to be robust and adaptable in continuing to manage and enforce policy in the face of attack or failure of any combination of components. Finally, KAoS addresses 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 sophisticated graphical user interfaces for monitoring, visualizing, and dynamically modifying policies at runtime.

2.2 KAoS Policy Ontologies

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. For an assessment of current description-logic-based representations and tools for policy based on our experience with KAoS, see [33]; for a comparison between the KAoS, Rei, and Ponder approaches to policy management, see [32].

Overview of DAML and KPO. The KAoS Policy Ontologies (KPO) are currently expressed in DAML (http://www.daml.org). Designed to support the emerging "Semantic Web„ 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 services and control mechanisms. OWL, a W3C-approved successor to DAML (http://www.w3.org/2001/sw/WebOnt), is currently being finalized and will be adopted in KAoS as soon as needed tools are in place.

The current version of KPO defines basic ontologies for actions, actors, groups, places, various entities related to actions (e.g., computing resources), and policies. There are currently about 80 classes and 40 properties defined in the basic ontologies. It is expected that for a given application, developers will further extend KPO. As the application runs, classes and individuals corresponding to new policies and application entities are also transparently added and deleted as needed.

Actors, actions, groups, and places. The actor ontology distinguishes between people and various classes of artificial

[2] In an insightful essay, Roger Clarke explores some of the implications of Asimov's stories about the laws of robotics for information technologists [10]. Weld and Etzioni [35] were the first to discuss the implications of Asimov's first law of robotics for agent researchers. Like most norm-based approaches described below (and unlike most policy-based approaches) the safety conditions are taken into account as part of the agents' own learning and planning processes rather than as part of the infrastructure. In an important response to Weld and Etzioni's "call to arms„ Pynadath and Tambe [23] develop a hybrid approach that marries the agents' probabilistic reasoning about adjustable autonomy with hard safety constraints to generate "policies„ governing the actions of agents. The approach assumes a set of homogeneous agents who are motivated to cooperate and follow optimally-generated policies.

[3] While it is true that over time norms can be formalized into laws, policies are explicit and formal by their very nature at the outset.

agents. Most agents are only permitted to perform *ordinary actions*, however various agents that are part of the infrastructure as well as authorized human users may variously be permitted or obligated to perform certain *policy actions,* such as policy approval and enforcement. Groups of actors or other entities may be distinguished according to whether the set of members is defined extensionally (i.e., through explicit enumeration in some kind of registry) or intensionally (i.e., by virtue of some common property such as a joint goal that all actors possess or a given place where various entities may temporarily or permanently be located).

Policies. 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) [13]. A policy is represented as a DAML instance of the appropriate policy type with associated values for properties: priority, update time stamp and a site of enforcement. The most imported property value is the name of a controlled action class. In most cases a new action class is built automatically whenever a policy is defined. 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 a particular group, or to agents running in a given physical place or computational environment. Additional aspects of the action context can be precisely described by restricting values of its properties.

The policy example below, drawn from the DARPA CoAX experiment (described in section 3), stipulates that the members of a domain named Arabello-HQ are forbidden to communicate with those outside this domain using unencrypted communication:[4]

```
<daml:Class rdf:ID="P1Action">
        <rdfs:subClassOf rdf:resource="#CommunicationAction" />
        <rdfs:subClassOf>
                <daml:Restriction>
                        <daml:onProperty
                rdf:resource="#performedBy" />
                        <daml:toClass
                rdf:resource="#MembersOfDomainArabello-HQ" />
                </daml:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
                <daml:Restriction>
                        <daml:onProperty
                        rdf:resource="#hasDestination"  />
                        <daml:toClass
                        rdf:resource="#notMembersOfDomain
                        Arabello-HQ" />
                </daml:Restriction>
        </rdfs:subClassOf>
</daml:Class>

<policy:NegAuthorizationPolicy rdf:ID="P1">
        <policy:controls rdf:resource="#P1Action" />
        <policy:hasSiteOfEnforcement rdf:resource="#ActorSite" />
        <policy:hasPriority>1</policy:hasPriority>
        <policy:hasUpdateTimeStamp>446744445544</policy:hasUp
dateTimeStamp>
</policy:NegAuthorizationPolicy
```

2.3 Define Policies and Domains with KPAT

KPAT provides a graphical user interface for specifying and modifying policies and domains.[5] In addition, KPAT can be used to browse and load ontologies and to deconflict newly defined policies. As policies, domains, and application entities are defined using KPAT, the appropriate DAML representations are generated automatically in the background and asserted into or retracted from the system, insulating the user from having to know DAML or any other policy language. A generic DAML policy editor may be used for this purpose (see figure 5 below). Specialized policy templates can also be defined to allow various classes of policy definitions to be defined as high-level domain-specific abstractions. A rich set of queries is also available through KPAT or through programmatic interfaces.

Groups of agents are structured into agent domains and subdomains to facilitate policy administration. Domains may represent any sort of group imaginable, from potentially complex organizational structures to administrative units to dynamic task-oriented teams with continually changing membership. A given domain can extend across host boundaries and, conversely, multiple domains can exist concurrently on the same host. Domains may be nested indefinitely and, depending on whether policy allows, agents may become members of more than one domain at a time.

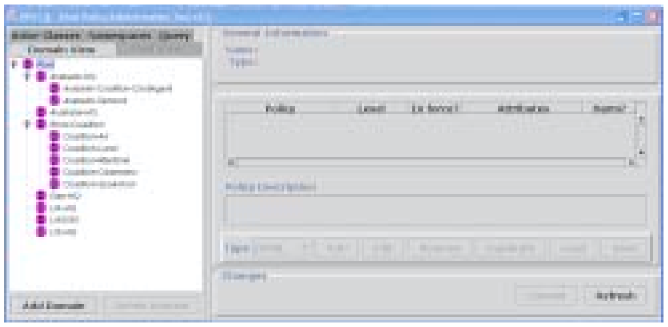


Figure 1. KPAT with the domain view showing multiple nested domains.

2.4 Policy Conflict Resolution

The KAoS Policy Ontologies are used for various forms of online or offline inference and analysis, including query-based policy disclosure management, 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.

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) or some other entity may require logical inference to determine first of all which policies are in conflict and second how to resolve these conflicts [21]. We have implemented a general-purpose algorithm for policy conflict detection and harmonization whose initial results promise a high degree of efficiency and scalability.

Figure 2 shows the three types of conflict that can currently be handled: positive vs. negative authorization (i.e., being simultaneously permitted and forbidden from performing some action), positive vs. negative obligation (i.e., being both required and not required to perform some action), and positive obligation vs. negative authorization (i.e., being required to perform a forbidden action). The use of policy deconfliction and harmonization algorithms that incorporate subsumption-based reasoning means that policy conflicts can be detected and resolved even when the actors, actions, or targets of the policies are specified at very different levels of abstraction. The policy conflict resolution algorithms rely on a version of Stanford's Java Theorem Prover (http://www.ksl.stanford.edu/software/JTP/) combined with our own KAoS-specific reasoning and query extensions.

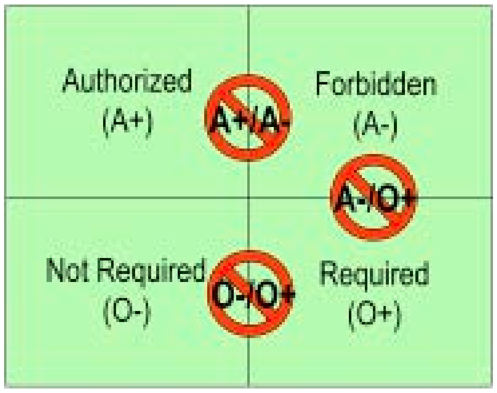
Steps in policy conflict resolution. KAoS performs several steps in order to resolve policy conflicts:

1. A DAML policy conflict ontology must be loaded into JTP along with the set of DAML policies to be deconflicted.
2. A Java list of all policies is constructed and sorted according to user-defined criteria for policy precedence.[6]

[4] Of course the DAML policy is not meant to be written or analyzed directly by an administrator; instead the KPAT user interface would be used to hide the complexity of the underlying representations

[5] Policies can also be defined, analyzed, or modified programmatically by trusted software components.

[6] We currently rely on numeric policy priority assignments by users to determine precedence. In the future we intend to allow people complete flexibility in designing the nature and scope of precedence conditions. For example, it would



3. For each policy in the sorted list, iterate through all the elements with a lower precedence and check to see if there is a policy conflict. A policy conflict occurs if the two policies are instances of conflicting types and if the JTP subsumption mechanism determines that the actions (comprising the action itself along with the actor and other entities associated with the action) that the two policies control are not disjoint.
4. The lower precedence policy from the conflicting pair of policies is removed from the Java list and the policy harmonization algorithm is invoked. It attempts to modify the policy with the lower precedence to the minimum degree necessary to resolve the conflict. If precedence cannot be determined otherwise, KAoS will ask the administrator to determine the appropriate action: either removing the policy, changing precedence, splitting the policy, or continuing with harmonization [33]. The harmonization algorithm may generate zero, one or several new policies to replace the removed policy.
5. The newly constructed harmonized policies inherit the precedence and the time of last update from the removed policy, and a pointer to the original policy is maintained so that it can be recovered if necessary as policies continue to be added or deleted in the future.

Details of policy harmonization. The derivation of the newly-generated set of harmonized policies from the original policies (P1 and P4) can be understood by imagining an intersection of two N-dimensional Cartesian products:

If

P1 and P4 are two Cartesian products[7] defined as:

$P1 = D11 \times D12 \times \ldots \times D1n$

$P4 = D21 \times D22 \times \ldots \times D2n$

then

$P1 \setminus P4 = subP1 + subP2 + \ldots + subPn$

where

$subPk =$
$(D11 \cap D21) \times \ldots \times (D1(k-1) \cap D2(k-1)) \times (D1k \setminus D2k) \times D1(k+1) \times .. \times D1n$

be possible to define precedence based on the relative authorities of the individual who defined or imposed the policies in conflict, which policy was defined first, which has the largest or smallest scope, whether negative or positive authorization trumps by default, whether subdomains takes precedence over superdomains or vice versa, etc.

[7] A Cartesian product is the collection of all ordered n-tuples that can be formed so that they contain one element of the first set, one element of the second, and so forth until you reach the *n*th set. This collection can be seen as constituting an n-dimensional space in which each n-tuple designates a cell.

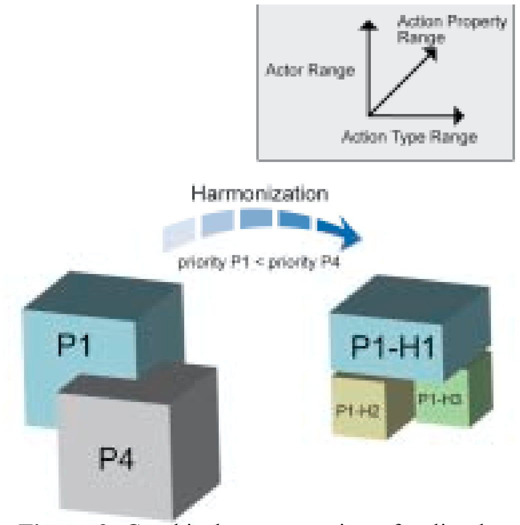


Figure 3. Graphical representation of policy harmonization.

Figure 3 shows a 3-D graphical representation of policy harmonization. The illustration, based on the example described in section 2.7 below, contains only a single action property. Mapping the mathematical definition above to the generation of harmonized policies we get the following:

1. The first harmonized policy has a range of actors that corresponds to the difference between the ranges of the two original policies and a controlled action and range of values on the action properties that correspond to those of the lower-precedence policy.
2. The second harmonized policy has a range of actors that corresponds to the intersection of the ranges of the two original policies, a controlled action that corresponds to the differences between those of the two policies, and a range of values on the action properties that correspond to that of the lower-precedence policy.
3. Additional harmonized policies are built to correspond to each action property in the two original policies. The range of actors corresponds to the intersection of the ranges of the two original policies and the controlled action corresponds to the intersection between those of the two policies.

The results of computing any of the above policies may be empty, in which case the result can be discarded. Recently, we have modified KAoS conflict resolution handling to obviate the need for harmonization in many cases, further increasing performance.

2.5 Policy Distribution

Figure 4 shows the major components of KAoS policy and domain services framework. During the initialization process, the core policy ontologies are loaded into the KAoS Directory Service (DS) using the namespace management capabilities of KPAT. Additional application-specific or platform-specific ontologies then can be loaded dynamically from KPAT or programmatically using the appropriate Java method. As the end-user application executes, instances relating to application entities are added and deleted as appropriate. For specific applications and platforms, the KAoS framework can be further extended and specialized by creating plug-ins for [33]:

- Policy template and custom action property editors;
- Enforcers controlling, monitoring, or facilitating general or specific actions;
- Classifiers to determine if a given instance is in the scope of the given class.

The DS implements domain management functionality, determining, for example, whether agents can join their domain and analyzing or deconflicting policies as required.

The DS is responsible for notifying Guards about changes in policy or other aspects of system state that may affect their operation.

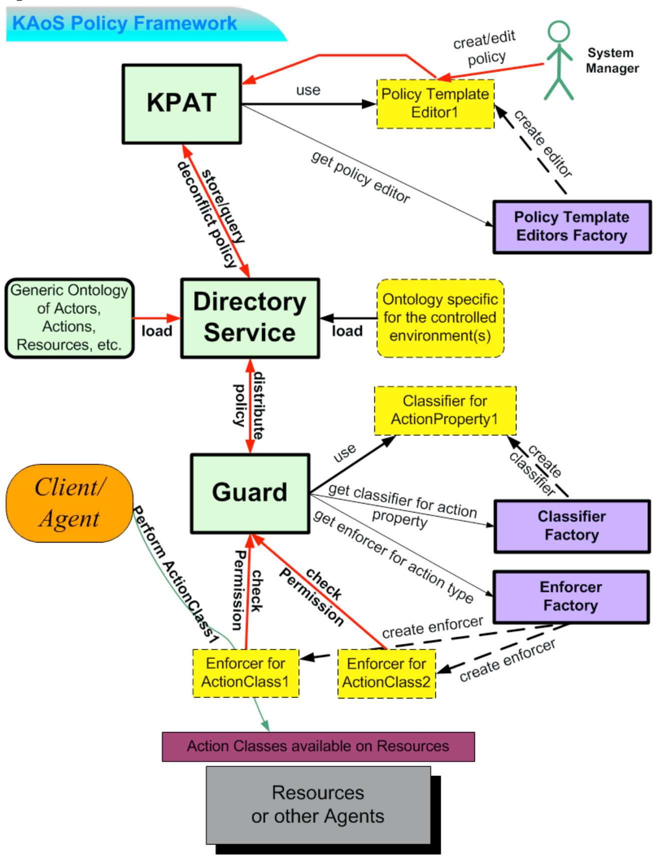

KAoS Policy Framework
creat/edit policy
System Manager
KPAT
use
Policy Template Editor1
create editor
get policy editor
Policy Template Editors Factory
store/query deconflict policy
Generic Ontology of Actors, Actions, Resources, etc.
load
Directory Service
load
Ontology specific for the controlled environment(s)
distribute policy
Classifier for ActionProperty1
use
create classifier
Guard
get classifier for action property
Classifier Factory
get enforcer for action type
Enforcer Factory
Client/ Agent
Perform ActionClass1
Permission check
check Permission
create enforcer
create enforcer
Enforcer for ActionClass1
Enforcer for ActionClass2
Action Classes available on Resources
Resources or other Agents

Figure 4. KAoS policy and domain services architecture.

Following conflict detection, policies are distributed to guards based on information about types of agents controlled by them. Guards activate appropriate enforcers based on received policy types. While KPAT, the DS, and the Guards are intended to work identically across different agent platforms (e.g., DARPA CoABS Grid, Cougaar, CORBA) and execution environments (e.g., Java VM, Aroma VM), enforcement mechanisms are typically designed for a specific platform and execution environment. Our approach enables policy uniformity in domains that might be simultaneously distributed across multiple platforms and execution environments, as long as semantically equivalent monitoring and enforcement mechanisms are available.

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.

2.6 Policy Enforcement

Enforcers are the mechanism by which Guards ensure compliance with authorization or obligation policies. The grounding of enforcers to platforms and environments cannot always be made fully generic. However, they can often be made fully general and understand abstract ontology action classes via their property `implementedBy` (which maps them to concrete environment operations) and through the use of reflection and security mechanism of the environment. Other environments require pre-building enforcers based on the ontology description of the controlled action class, potentially using a preprocessor. Finally, some cases required fully custom built enforcers. What can be made generic however is the interface to the policy disclosure system answering the question, in the case of authorization policies, "Is a given action authorized or not?„[8]

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 in the past developed methods based on JAAS that 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.

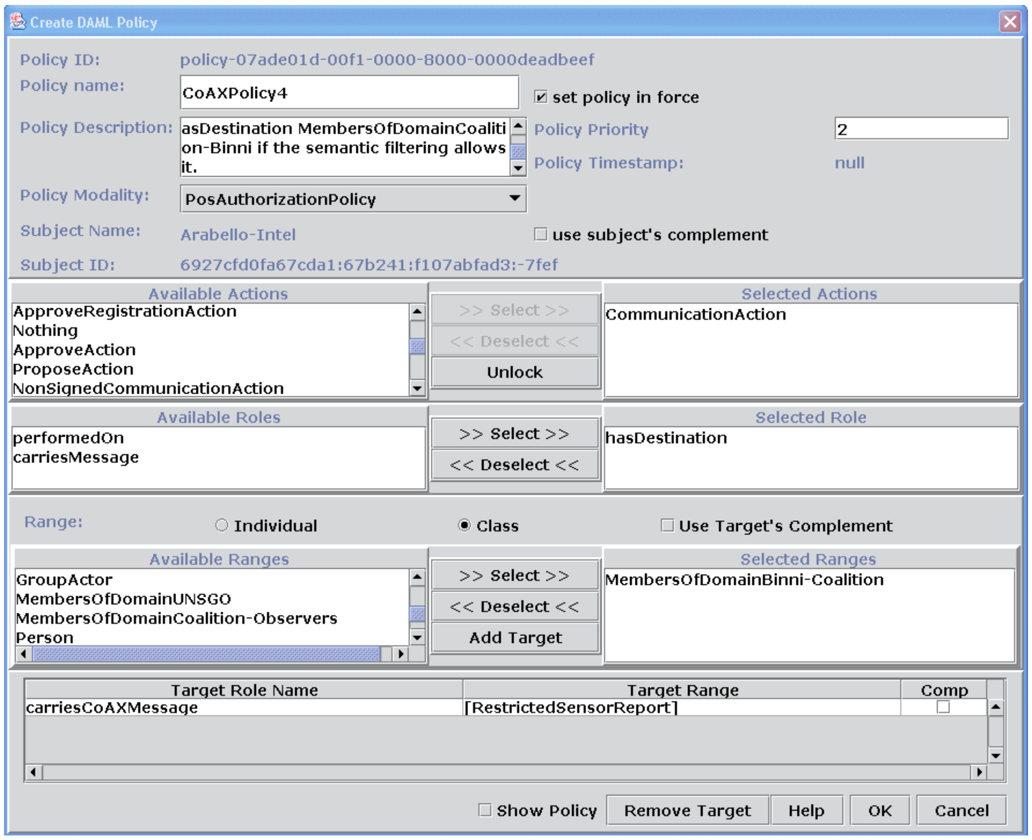
A second kind of enforcement mechanism takes the form of extensions to particular agent platform capabilities. Agents that participate in that platform are generally given more permissions to the degree they are able to make small adaptations in their agents to comply with policy requirements. For example, 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 usually 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 to its supervisor every five 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). In addition to enforcers that monitor the performance of obligations, a second type of enforcer called an *enabler* goes beyond simple monitoring to proactively facilitate or perform the obligation on behalf of the agent. For example, a monitor might not only watch whether the agent described above reports every five minutes, but actively facilitate the fulfillment of its obligation by querying its status every five minutes and making the report to its supervisor on its behalf.[9]

Each policy has a property that defines the site of policy enforcement. For example, access control policies are typically enforced by a mechanism directly associated with the resource to be protected (i.e., the *target*). However in some cases, administrators may not have control over this resource and instead may require the policy to be enforced by a mechanism associated with the actor (i.e., the *subject*) or some other entity under their purview.

[8] To better support policy exploration we are implementing a variety of additional policy disclosure mechanisms to help users or framework components answer various "what if„ and "how to„ questions, e.g., test permission, get obligations, learn options, test alternatives, or get consequences.

[9] *Enablers* can also be used in conjunction with certain kinds of authorization policies.


Create DAML Policy

Policy ID: policy-07ade01d-00f1-0000-8000-0000deadbeef

Policy name: CoAXPolicy4 ☑ set policy in force

Policy Description: asDestination MembersOfDomainCoalition-Binni if the semantic filtering allows it.

Policy Priority: 2

Policy Timestamp: null

Policy Modality: PosAuthorizationPolicy

Subject Name: Arabello-Intel ☐ use subject's complement

Subject ID: 6927cfd0fa67cda1:67b241:f107abfad3:-7fef

Available Actions		Selected Actions
ApproveRegistrationAction	>> Select >>	CommunicationAction
Nothing	<< Deselect <<	
ApproveAction	Unlock	
ProposeAction		
NonSignedCommunicationAction		

Available Roles		Selected Role
performedOn	>> Select >>	hasDestination
carriesMessage	<< Deselect <<	

Range: ○ Individual ● Class ☐ Use Target's Complement

Available Ranges		Selected Ranges
GroupActor	>> Select >>	MembersOfDomainBinni-Coalition
MembersOfDomainUNSGO	<< Deselect <<	
MembersOfDomainCoalition-Observers	Add Target	
Person		

Target Role Name	Target Range	Comp
carriesCoAXMessage	[RestrictedSensorReport]	☐

☐ Show Policy | Remove Target | Help | OK | Cancel

2.7 Policy Example

To better explain policy conflict resolution we will describe a simple English-language example of the process and results. The example is taken from the Coalition Agents Experiment (CoAX) described below (section 3).

As part of the CoAX scenario, the fictitious country of Arabello joined the coalition. One interaction involved a coalition agent tasked to locate a hostile submarine and an Arabello Intel agent capable of providing sensor reports from an underwater sensor grid. As new coalition partners, Arabello system administrators dynamically allowed sensor contact reports to be sent to the coalition agent, but for security reasons, restricted the range of messages that could be sent outside of the Arabello domain. The limitation, described as part of a semantic filtering policy [28; 29] represented in DAML, limited these outgoing messages to those whose content was reports about a specific class of submarine, belonging to the enemy forces, but disallowing reports on other ships, such as those of Arabello itself.

A global default positive authorization policy for the entire coalition was previously decided:

> *P0: **Allow coalition actors to perform any action that is not explicitly prohibited by policy**.*[10]

The coalition could have just as easily implemented a negative authorization policy as a default, prohibiting any action that was not explicitly authorized by policy.

Arabello headquarters decides on the following restrictive default policy for the actors in their domain:

> *P1: Negative Authorization on MembersOfDomainArabello-HQ to perform CommunicationAction on hasDestination complementOf MembersOfDomainArabello-HQ. (i.e., **Prohibit outgoing communication between members of the Arabello domain and any actor outside the Arabello domain**.)*

However Arabello-Contingent administrators would like to enable the Arabello Intel agent to be able to send a subset of its reports to the coalition. It defines the following policy, which is allocated a higher priority than the previous policy (figure 5):

> *P4: Positive Authorization on Arabello-Intel to perform CommunicationAction on hasDestination MembersOfDomainCoalition-Binni if the semantic filter allows it (i.e., **Allow the Arabello Intel agent to send outgoing messages about enemy submarines to members of the Binni-Coalition domain**).*

When the Arabello administrators commit policies P1 and P4, KAoS first identifies the policy conflict inherent in the fact that the two policies are mutually inconsistent (i.e., P1 disallows any communication outside Arabello while P4 permits selected communication). Since P4 was defined to be of higher priority, it remains in force unchanged while P1 becomes the subject for policy harmonization. The result is three new harmonized policies, all with the same priority of the original P1:

> *P1-H1: Negative Authorization on MembersOfDomainArabello-HQ difference Arabello-Intel to perform CommunicationAction on hasDestination (complementOf MembersOfDomainArabello-HQ) (i.e., **Prohibit outgoing communication for all Arabello domain members except the Arabello Intel agent to members of non-Arabello domains**).*

[10] Default authorization modalities are currently configured on a per domain basis. The defaults either correspond to a *democracy,* where everything is permitted that is not explicitly forbidden, or a *tyranny,* where everything is forbidden that is not explicitly permitted.

P1-H3: Negative Authorization on Arabello-Intel to perform CommunicationAction on hasDestination (complementOf MembersOfDomainArabello-HQ) difference MembersOfDomainBinni-Coalition (i.e., ***Prohibit outgoing communication by the Arabello Intel agent to any actor that is not a member of the Binni-Coalition domain****).*

P1-H4: Negative Authorization on Arabello-Intel to perform CommunicationAction on hasDestination (complementOf MembersOfDomainArabello-HQ) intersection MembersOfDomainBinni-Coalition if the semantic filter does not allow it (i.e., ***Prohibit outgoing communication by the Arabello Intel agent to any actor outside the Arabello domain who is a member of the Binni-Coalition domain if the semantic filter does not allow it****).*

The first policy (P1-H1) corresponds to the first type of harmonized policy described in section 2.4 and shown in Figure 3; the other two policies (P1-H3 and P1-H4) correspond to the third type of harmonized policy. Since both policies constrained the identical class of action, no policy of the second type was generated.

Following harmonization, the user is notified and given an opportunity to resolve any remaining issues and approve the results of policy conflict resolution (figure 6). Following user approval, any obsolete policies are removed and new policies are sent to the appropriate enforcers. In this case, a communication enforcer associated with the Arabello Intel agent is requested to remove P1 and replace it with P1-H3 and P1-H4. Policy P4 is also sent to this enforcer. The next version of the policy distribution mechanism will take information about the enforcers default behavior into account and will not distribute policies in such a case. Communication enforcers associated with each of the other agents in the Arabello domain are requested to remove P1 and replace it with P1-H1.

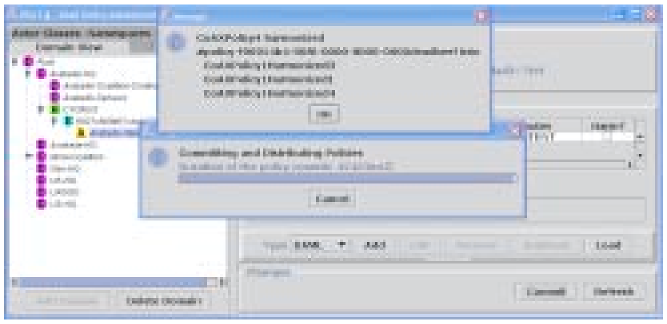


Figure 6. KPAT notifies the user of the results of policy harmonization and any issues that have arisen.

Performance. We have tested the performance of KAoS policy conflict resolution algorithms on a machine with Pentium III 1.2 GHz and 640 MB RAM using JDK 1.3.1. In the limited non-optimized tests we have made to date, policy commitment, conflict resolution, and harmonization is consistently performed in a fraction of a second. For reasons that are not yet fully understood, however, assertion of each new policy into the JTP database typically takes an order of magnitude longer than that. For this reason, we have recently implemented a workaround that does not require policy instances to be represented in JTP. Stanford JTP developers are also working on performance improvements.

3. APPLICATIONS

Research and development of KAoS and Nomads is taking place in the context of several applications.

The DARPA CoABS-sponsored Coalition Operations Experiment (CoAX) (http:// www.aiai.ed.ac.uk /project/ coax/) [2; 28] is a large international cooperation that models military coalition operations and implements agent-based systems to mirror coalition structures, policies, and doctrines. CoAX 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 operational constraints represented as policies, while Nomads provides strong mobility, resource management, and protection from denial-of-service attacks for untrusted agents that run in its environment.

Within the DARPA Ultra*Log program (http://www.ultralog.net) we are collaborating with Network Associates (NAI) to extend and apply KAoS policy and domain services to assure the scalability, robustness, and survivability of logistics functionality in the face of information warfare attacks or severely constrained or compromised computing and network resources.

As part of the Army Research Lab Advanced Decision Architectures Consortium, we have been investigating the use of KAoS and Nomads technologies to enable soldiers in the field to use agents from handheld devices to perform tasks such as dynamically tasking sensors and customizing information retrieval [29; 31]. Suri has developed an agile computing platform [30] that provides a foundation for this work. We have also commenced an investigation of requirements for policy-based information access and analysis within intelligence applications.

An application focused more on the social aspects of agent policy is within the NASA Cross-Enterprise and Intelligent Systems Programs, where we are investigating the use of policy-based models to drive human-robotic teamwork and adjustable autonomy for highly-interactive autonomous systems such as the Personal Satellite Assistant (PSA), a softball-sized flying robot that is being designed to operate onboard spacecraft in pressurized micro-gravity environments [1; 8]. The same approach is also being generalized for use in other testbeds, such as unmanned vehicles and other highly interactive autonomous systems [25].

The Office of Naval Research (ONR) is supporting research to extend this work on effective human-agent interaction to unmanned vehicles and other autonomous systems that involve close, continuous interaction with people. As one part of this research IHMC and University of South Florida are developing a new robotic platform with carangiform (fish-like) locomotion, specialized robotic behaviors for humanitarian demining, human-agent teamwork, agile computing, and mixed-initiative human control.

Under funding from DARPA's Augmented Cognition Program, we are taking the challenge of effective human-agent interaction one step further as we investigate whether a general policy-based approach to the development of cognitive prostheses can be formulated, in which human-agent teaming could be so natural and transparent that robotic and software agents could appear to function as direct extensions of human cognitive, kinetic, and sensory capabilities [15; 16].

4. FUTURE DIRECTIONS

Future work will include: performance enhancements to reasoning mechanisms, simplification and streamlining of the KPAT user interface, and policy implementation constraint resolution to deal with contention for finite resources.

5. ACKNOWLEDGMENTS

The authors gratefully acknowledge the sponsorship of this research by the NASA Cross-Enterprise and Intelligent Systems Programs, and a joint NASA-DARPA ITAC grant. Additional support was provided by DARPA's CoABS, Ultra*Log, and Augmented Cognition programs, by the Army Research Lab's Advanced Decision Architectures program (ARLADA), and the Office of Naval Research. We are also grateful for the contributions of Patrick Beautement, Guy Boy, Thomas Bradshaw, Marshall Brinn, Murray Burke, Bill Clancey, Rob Cranfill, Paul Feltovich, Richard Fikes, Ken Ford, Mark Greaves, Jack Hansen, Robert Hoffman, Wayne Jansen, Jessica Jenkins, Mike Kerstetter, Deborah McGuinness, Sheila McIlraith, Nicola Muscettola, Anil Raj, Timothy Redmond, Sue Rho, Dylan Schmorrow, Mike Shafto, Morris Sloman, Austin Tate, Milind Tambe, and Tim Wright.

6. REFERENCES

[1] Acquisti, A., Sierhuis, M., Clancey, W. J., & Bradshaw, J. M. (2002). Agent-based modeling of collaboration and work practices onboard the International Space Station. *Proceedings of the Eleventh Conference on Computer-Generated Forces and Behavior Representation*. Orlando, FL,

[2] 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*, 17(3), 26-35.

[3] Asimov, I. (1942/1968). Runaround. In I. Asimov (Ed.), *I, Robot.* (pp. 33-51). London, England: Grafton Books. Originally published in *Astounding Science Fiction,* 1942, pp. 94-103.

[4] Boman, M. (1999). Norms in artificial decision-making. *Artificial Intelligence and Law*, 7, 17-35.

[5] Bradshaw, J. M., Beautement, P., Raj, A., Johnson, M., Kulkarni, S., & Suri, N. (2003). Making agents acceptable to people. In N. Zhong & J. Liu (Ed.), *Intelligent Technologies for Information Analysis: Advances in Agents, Data Mining, and Statistical Learning.* (pp. in press). Berlin: Springer Verlag.

[6] 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.

[7] 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.

[8] Bradshaw, J. M., Sierhuis, M., Acquisti, A., Feltovich, P., Hoffman, R., Jeffers, R., Prescott, D., Suri, N., Uszok, A., & Van Hoof, R. (2003). Adjustable autonomy and human-agent teamwork in practice: An interim report on space applications. In H. Hexmoor, R. Falcone, & C. Castelfranchi (Ed.), *Agent Autonomy.* (pp. in press). Kluwer.

[9] Bradshaw, J. M., Suri, N., Breedy, M. R., Canas, A., Davis, R., Ford, K. M., Hoffman, R., Jeffers, R., Kulkarni, S., Lott, J., Reichherzer, T., & Uszok, A. (2002). Terraforming cyberspace. In D. C. Marinescu & C. Lee (Ed.), *Process Coordination and Ubiquitous Computing.* (pp. 165-185). Boca Raton, FL: CRC Press. Updated and expanded version of an article that originally appeared in IEEE Intelligent Systems, July 2001, pp. 49-56.

[10] Clarke, R. (1993-1994). Asimov's laws of robotics: Implications for information technology, Parts 1 and 2. *IEEE Computer*, December/January, 53-61/57-66.

[11] Conte, R., & Castelfranchi, C. (1995). *Cognitive and social action*. London, England: UCL Press.

[12] d'Inverno, M., & Luck, M. (2001). *Understanding Agent Systems*. Berlin, Germany: Springer-Verlag.

[13] 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.

[14] Feltovich, P., Bradshaw, J. M., Jeffers, R., & Uszok, A. (2003). Order and KAoS: Using policy to represent agent cultures. *Proceedings of the AAMAS 03 Workshop on Humans and Multi-Agent Systems*. Melbourne, Australia,

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

[16] Hoffman, R. R., Ford, K. M., Hayes, P. J., & Bradshaw, J. M. (2003). The Borg hypothesis. *IEEE Intelligent Systems*, in press.

[17] Johnson, M., Chang, P., Jeffers, R., Bradshaw, J. M., Soo, V.-W., Breedy, M. R., Bunch, L., Kulkarni, S., Lott, J., Suri, N., & Uszok, A. (2003). KAoS semantic policy and domain services: An application of DAML to Web services-based grid architectures. *Proceedings of the AAMAS 03 Workshop on Web Services and Agent-Based Engineering*. Melbourne, Australia,

[18] Kahn, M., & Cicalese, C. (2001). CoABS Grid Scalability Experiments. O. F. Rana (Ed.), *Second International Workshop on Infrastructure for Scalable Multi-Agent Systems at the Fifth International Conference on Autonomous Agents*. Montreal, CA, New York: ACM Press,

[19] Lopez y Lopez, F., Luck, M., & d'Inverno, M. (2001). A framework for norm-based inter-agent dependence. *Proceedings of the Third Mexican Internation Conference on Computer Science*.

[20] Lopez y Lopez, F., Luck, M., & d'Inverno, M. (2002). Constraining autonomy through norms. *Proceedings of the Conference on Autonomous Agents and Multi-Agent Systems,* (pp. 674-681). Bologna, Italy,

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

[22] Norman, D. A. (1997). How might people interact with agents? In J. M. Bradshaw (Ed.), *Software Agents.* (pp. 49-55). Cambridge, MA: The AAAI Press/The MIT Press.

[23] Pynadath, D., & Tambe, M. (2001). Revisiting Asimov's first law: A response to the call to arms. *Proceedings of ATAL 01.*

[24] Shoham, Y., & Tennenholtz, M. (1992). On the synthesis of useful social laws for artificial agent societies. *Proceedings of the Tenth National Conference on Artificial Intelligence,* (pp. 276-281). San Jose, CA,

[25] Sierhuis, M., Bradshaw, J. M., Acquisti, A., Van Hoof, R., Jeffers, R., & Uszok, A. (2003). Human-agent teamwork and adjustable autonomy in practice. *Proceedings of the Seventh International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*. Nara, Japan,

[26] 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,

[27] 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,

[28] Suri, N., Bradshaw, J. M., Burstein, M. H., Uszok, A., Benyo, B., Breedy, M. R., Carvalho, M., Diller, D., Groth, P. T., Jeffers, R., Johnson, M., Kulkarni, S., & Lott, J. (2003). DAML-based policy enforcement for semantic data transformation and filtering in multi-agent systems. *Proceedings of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2003)*. Melbourne, Australia, New York, NY: ACM Press,

[29] Suri, N., Bradshaw, J. M., Carvalho, M., Breedy, M. R., Cowin, T. B., Saavendra, R., & Kulkarni, S. (2003). Applying agile computing to support efficient and policy-controlled sensor information feeds in the Army Future Combat Systems environment. *Proceedings of the Annual U.S. Army Collaborative Technology Alliance (CTA) Symposium*.

[30] Suri, N., Bradshaw, J. M., Carvalho, M., Cowin, T. B., Breedy, M. R., Groth, P. T., & Saavendra, R. (2003). Agile computing: Bridging the gap between grid computing and ad-hoc peer-to-peer resource sharing. O. F. Rana (Ed.), *Proceedings of the Third International Workshop on Agent-Based Cluster and Grid Computing*. Tokyo, Japan,

[31] Suri, N., Carvalho, M., Bradshaw, J. M., Breedy, M. R., Cowin, T. B., Groth, P. T., Saavendra, R., & Uszok, A. (2003). Mobile code for policy enforcement. *Policy 2003*. Como, Italy,

[32] Tonti, G., Bradshaw, J. M., Jeffers, R., Montanari, R., Suri, N., & Uszok, A. (2003). Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. *Submitted to the International Semantic Web Conference (ISWC 03)*. Sanibel Island, Florida,

[33] Uszok, A., Bradshaw, J. M., Hayes, P., Jeffers, R., Johnson, M., Kulkarni, S., Breedy, M. R., Lott, J., & Bunch, L. (2003). DAML reality check: A case study of KAoS domain and policy services. *Submitted to the International Semantic Web Conference (ISWC 03)*. Sanibel Island, Florida,

[34] Verhagen, H. (2001). Norms and artificial agents. *Sixth Meeting of the Special Interest Group on Agent-Based Social Simulation, ESPRIT Network of Excellence on Agent-Based Computing*. Amsterdam, Holland, http://abss.cfpm.org/amsterdam-01/abssnorms.pdf.

[35] Weld, D., & Etzioni, O. (1994). The firsts law of robotics: A call to arms. *Proceedings of the National Conference on Artificial Intelligence (AAAI 94),* (pp. 1042-1047).