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 [16], the DARPA CoABS Grid [11], 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://www.java-agent.org). For a full description of KAoS, the reader is referred to [4; 5; 6; 7].

Nomads combines the capabilities of Aroma, an enhanced Java-compatible Virtual Machine (VM), with the Oasis agent execution environment [15]. 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 the means 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

Policy-based management approaches have grown considerably in popularity over the last few years, with increasing attention in the agent community to the Java programming language (http://java.sun.com/security/). 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 scope of our policy-based agent management approach includes the 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 [4; 10]. In addition to conversation policies, we are in the process of developing representations and enforcement mechanisms for *mobility policies [12], 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 policy enforcement mechanisms 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.

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. 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.

Overview of DAML and KPO. The KAoS Policy Ontologies (KPO) are expressed in DAML (http://www.daml.org). 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 services and control mechanisms.

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 79 classes and 41 properties defined in the basic ontologies. It is expected that for a given application, the ontologies will be further extended with additional classes, individuals, and rules.

Actors, actions, groups, and places. The actor ontology distinguishes between people and various classes of agents. Most agents can only perform *ordinary actions*, however various agents that are part of the infrastructure as well as authorized human user may variously be permitted or obligated to perform certain *policy actions,* such as policy

[1] Pronounced "KAY-pat."

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. A policy is a statement enabling or constraining execution of some type of action by one or more actors in relation to various aspects of some situation. 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]. Policies are represented with a DAML definition of the actor class defining the range of action instances that are controlled by this policy. Through various property restrictions in the action type, 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 (e.g., host, VM). Here is a policy example:

```
<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 Domain and Policy Specification using KPAT

The basic `policytool` Java currently provides assists users in editing Java 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 KAoS Policy Administration Tool (KPAT) implements a graphical user interface to policy management functionality. It has been developed to make policy specification, revision, and application easier for administrators without extensive training (figure 1). Using KPAT, an authorized user may make changes to agent policy from anywhere using a secure Web browser.[2] Alternatively, trusted infrastructure components such as Guards may, if authorized, propose policy changes autonomously or semi-autonomously based on their observation of system events.

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.

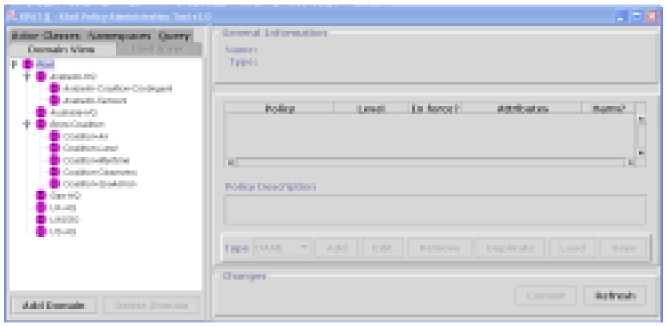


Figure 1. KPAT with the domain view showing multiple nested domains.

KPAT can also be used to browse and load ontologies, to define, deconflict, and commit new policies, and to modify or delete old ones. A generic DAML policy editor may be used for this purpose (see figure 5 below). Custom editors tailored to particular kinds of policy may also be accessed from a KPAT pop-up menu.

2.4 Policy Conflict Resolution

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. We expect to use the ontologies 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.

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 require logical inference to determine first of all which policies are in conflict and second how to resolve these conflicts [13]. We have implemented a general-purpose algorithm for policy conflict detection and harmonization whose initial results promise a high degree of efficiency and scalability.

[2] At the present time, a user can make changes to any of the domains, policies, or other DAML classes and instances once they have authenticated. In the future, access privileges will be differently scoped for different users.

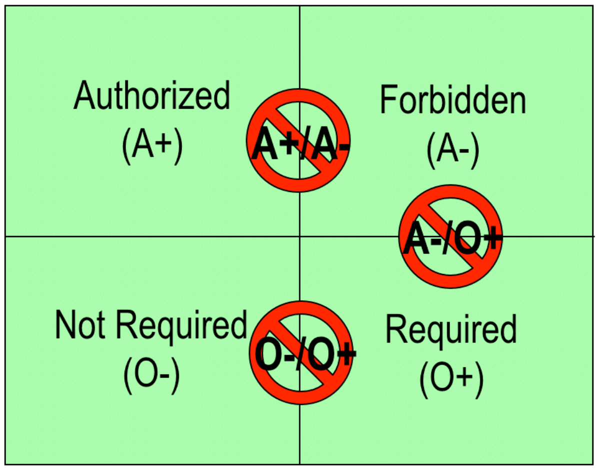


Figure 2 shows the three types of conflict that can currently be handled: positive vs. negative authorization (i.e., being simultaneously permitted and forbidden from performing some action), positive vs. negative obligation (i.e., being both required and not required to perform some action), and positive obligation vs. negative authorization (i.e., being required to perform a forbidden action). We have developed policy deconfliction and harmonization algorithms within KAoS to allow policy conflicts to be detected and resolved even when the actors, actions, or targets of the policies are specified at vastly different levels of abstraction. These algorithms rely in part on a version of Stanford's Java Theorem Prover (http://www.ksl.stanford.edu/software/JTP/) that we have integrated with KAoS.

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.[3]
3. For each policy in the sorted list, iterate through all the elements with a lower priority 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 priority 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 the policies are of equal precedence, a user may be required to specify which policy will take precedence). 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 can be understood by imagining an intersection of two N-dimensional Cartesian products:

If

P1 and P4 are two Cartesian products[4] defined as:

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

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

then

$P1 \backslash P2 = subP1 + subP2 + \ldots + subPn$

where

subPk =

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

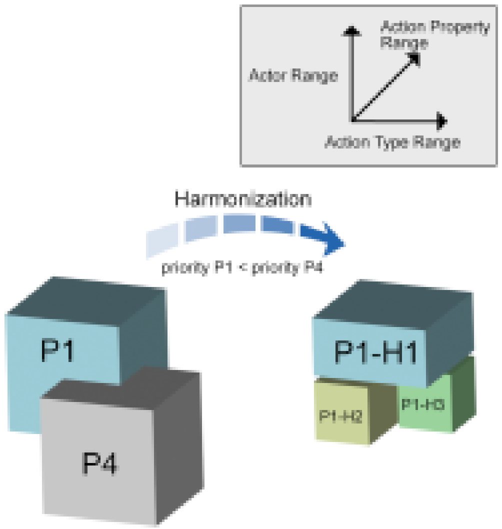


Figure 3. Graphical representation of policy harmonization.

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

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

[4] 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.

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.

2.5 Policy Distribution

Figure 4 shows the major components of KAoS policy and domain services architecture.

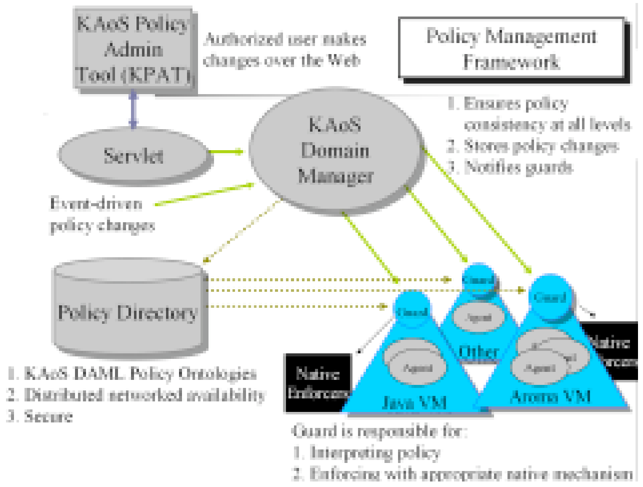

KAoS Policy Admin Tool (KPAT)
Authorized user makes changes over the Web
Policy Management Framework
Servlet
KAoS Domain Manager
1. Ensures policy consistency at all levels
2. Stores policy changes
3. Notifies guards
Event-driven policy changes
Policy Directory
1. KAoS DAML Policy Ontologies
2. Distributed networked availability
3. Secure
Guard
Agent
Other
Native Enforcer
Java VM
Aroma VM
Guard is responsible for:
1. Interpreting policy
2. Enforcing with appropriate native mechanism

Figure 4. KAoS policy and domain services architecture.

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.[5] 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.

[5] In the current implementation, DM's delegate inference about policy decisions to the directory service, which incorporates Stanford's Java Theorem Prover..

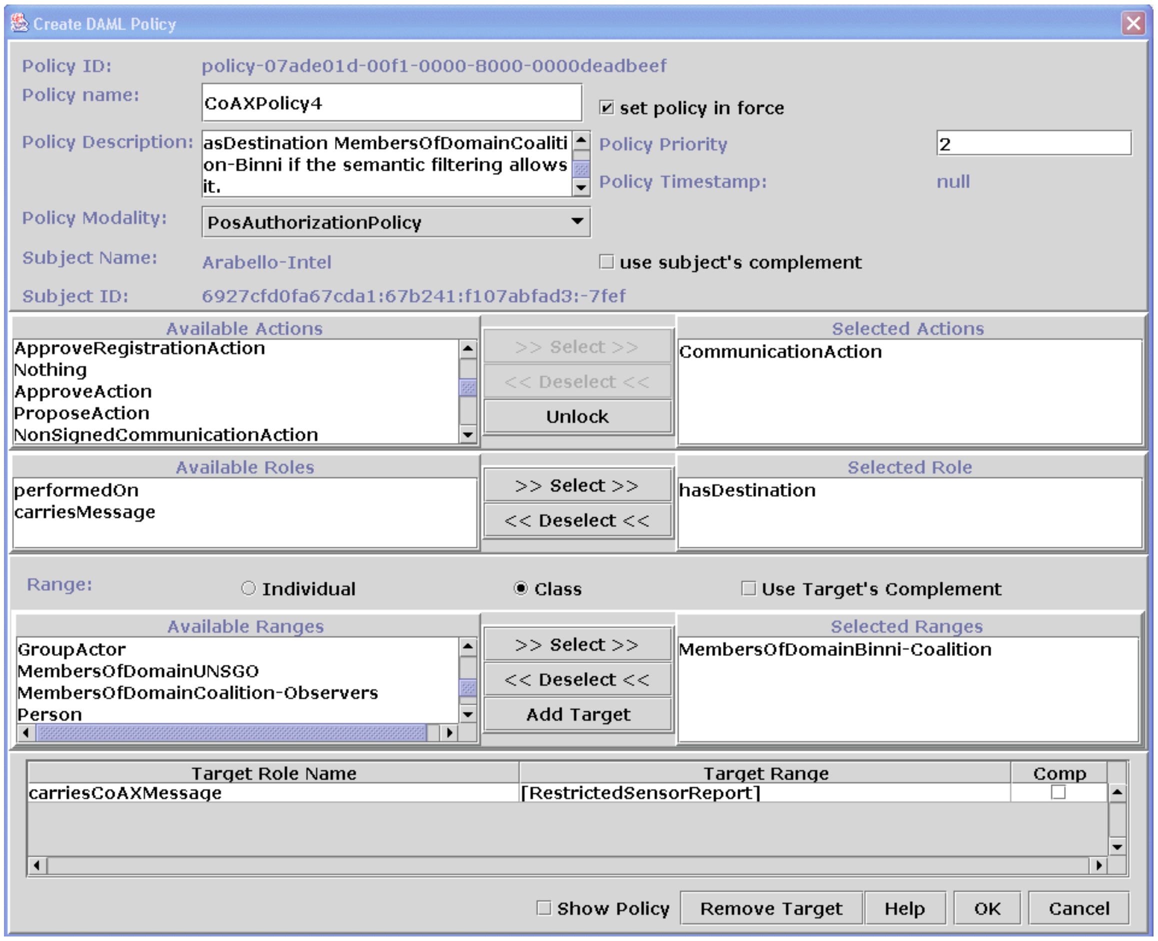
Policies are stored within ontologies in the directory service (DS). Although DM's normally provide the limited public interface to the DS, private interfaces may allow the DS to be accessed by other authorized entities in accordance with policy disclosure strategies [14]. For example, trusted agents 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 execution 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.

2.6 Policy Enforcement

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



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: ApproveRegistrationAction, Nothing, ApproveAction, ProposeAction, NonSignedCommunicationAction

Selected Actions: CommunicationAction

Available Roles: performedOn, carriesMessage

Selected Role: hasDestination

Range: ( ) Individual [● Class] [ ] Use Target's Complement

Available Ranges: GroupActor, MembersOfDomainUNSGO, MembersOfDomainCoalition-Observers, Person

Selected Ranges: MembersOfDomainBinni-Coalition

Target Role Name	Target Range	Comp
carriesCoAXMessage	[RestrictedSensorReport]	☐

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 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).

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.

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 3.1).

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 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.***[6]

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**).*

*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; 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.

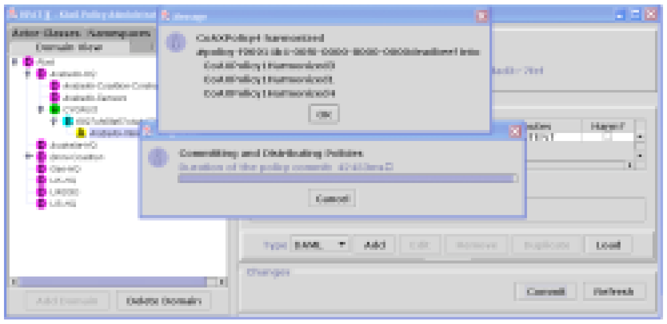


Figure 6. KPAT notifies the user of the results of policy harmonization and any issues that have arisen.

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. Stanford JTP developers are currently working on performance improvements that should significantly affect these results.

3. APPLICATIONS

KAoS and Nomads policy and domain services are being extended and evaluated in the context of several applications.

The first application is the DARPA CoABS-sponsored Coalition Operations Experiment (CoAX) (http:// www.aiai.ed.ac.uk /project/ coax/) [1; 17]. 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

[6] The global default policy is not explicitly represented in the ontology but is currently toggled by other means.

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.

Within the DARPA Ultra*Log program (http://www.ultralog.net) we are developing agent policy and domain services to assure the robustness and survivability of logistics functionality in the face of information warfare attacks or severely constrained or compromised computing and network resources.

Another application 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 [6]. The same approach is also being generalized for use in other testbeds, such as unmanned vehicles and other highly interactive autonomous systems.

Under funding from DARPA's Augmented Cognition Program, we are taking this approach one step further as we investigate whether a general policy-based approach to the development of cognitive prostheses can be formulated, where 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 [3; 9].

4. FUTURE DIRECTIONS

Future work will include: performance enhancements to reasoning mechanisms, simplification and streamlining of the KPAT user interface, obligation policies, 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, and by the Army Research Lab's Advanced Decision Architectures program (ARLADA). 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] 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.

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

[3] Bradshaw, J. M., Beautement, P., Raj, A., Johnson, M., Kulkarni, S., & Suri, N. (2002). Making agents acceptable to people. In N. Zhong & J. Liu (Ed.), *Handbook of Intelligent Information Technology*. (pp. in preparation). Amsterdam, The Netherlands: IOS Press.

[4] 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.

[5] 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.

[6] Bradshaw, J. M., Sierhuis, M., Acquisti, A., Feltovich, P., Hoffman, R., Jeffers, R., Prescott, D., Suri, N., Uszok, A., & Van Hoof, R. (2002). 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.

[7] 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. Expanded version of an article originally published in *IEEE Intelligent Systems*, July 2001, pp. 49-56.

[8] 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.

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

[10] 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.

[11] 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,

[12] 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,

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

[14] 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*.

[15] 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,

[16] 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,

[17] 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. (2002). DAML-based policy enforcement for semantic data transformation and filtering in multi-agent systems. (pp. submitted for publication).