Using KAoS Policy and Domain Services within Cougaar

James Lott, Jeffrey M. Bradshaw, Andrzej Uszok, Renia Jeffers
Institute for Human and Machine Cognition (IHMC), 40 S. Alcaniz St., Pensacola, FL 32502 USA
{jlott, jbradshaw, auszok, rjeffer}@ihmc.us

Abstract

KAoS policy and domain management services allow for the specification, management, conflict resolution, and enforcement of policies represented in OWL within contexts established by domains. We discuss the application of KAoS services in providing policy management for robustness and survivability in the context of the DARPA UltraLog program, a large-scale distributed agent-based system running on the Cougaar agent infrastructure. Over the course of the program we were able to demonstrate that a semantically-rich policy system could provide exceptional performance and responsiveness even under very demanding conditions.

1. Introduction

In 2000, the Defense Advanced Research Projects Agency (DARPA) initiated the UltraLog program (http://www.ultralog.net), a research effort focused on the development of technologies to enhance the security and robustness of large-scale, distributed agent-based systems operating in chaotic wartime environments. The objective of the UltraLog program is to create a comprehensive capability that will enable trusted, distributed agent infrastructure for operational logistics to be survivable under the most extreme circumstances, including direct attack and loss of significant pieces of system infrastructure. The prototype application is a logistics information system comprised of over a thousand agents of medium complexity, running on the Cougaar agent infrastructure.

In this paper, we discuss the application of KAoS policy and domain management services to the problem of policy-based control of security mechanisms in the context of the UltraLog program.

2. KAoS Policy and Domain Services

KAoS services and tools allow for the specification, management, conflict resolution, and enforcement of *policies* within contexts established as *domains* [1; 2; 3; 7; 9] . While initially oriented to the dynamic and complex requirements of software and robotic agent applications, KAoS services have been extended to work equally well with traditional clients on CORBA, Grid Computing, and Web Services platforms [5; 8].

Policies are declarative constraints on system behavior that provide a powerful means for dynamically regulating the behavior of components without changing code nor requiring the cooperation of the components being governed (http://www.policy-workshop.org/) [1]. The KAoS 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 when a state- or event-based trigger occurs, or else serve to waive such a requirement) [4]. Other policy constructs (e.g., delegation, role-based authorization) are built out of the basic primitives of domains plus these four policy types.

The concept of action is central to the definition of KAoS policy. Action is defined as ontological class used to classify instances of action that are intended or currently underway. If the action instance is of the action class type associated with the given policy then this policy is known to be applicable to the current situation.

The use of OWL enables reasoning about the controlled environment, policy relations and disclosure, policy conflict detection, and harmonization, as well as about domain structure and concepts exploiting description-logic-based subsumption and instance classification algorithms and, if necessary, controlled extensions to description logic (e.g., role-value maps). No rules are used in policy representation—rather conditions are expressed as property restrictions on actions associated with the policy ontologies.

A comparison of KAoS, Rei, and Ponder approaches to policy can be found in [6]. We highlight a few important features below.

Homogeneous representation. Because all aspects of KAoS representation are encoded purely in OWL, any third-party tool or environment supporting OWL can perform specialized analyses of the full knowledge base with complete independence from KAoS itself, thus easing integration with an increasingly sophisticated range of new OWL tools and language enhancements in the future.

Maturity. Over the past few years, KAoS has been used in a wide variety of applications and operating environments ranging from security for distributed systems to management of distributed sensors to policy-based interaction with software and robotic agents.

Comprehensiveness. Unlike many approaches that deal with only simple forms of access control or authorization, KAoS supports both authorization and obligation policies. In addition, a complete infrastructure for policy management has been implemented including a

full range of capabilities from sophisticated user interfaces for policy specification and analysis to generic policy enforcers. Facilities for policy enforcement automation (i.e., automatic generation of code for enforcers) are in development.

Pluggability. Platform-specific and application-specific ontologies are easily loaded on top of the core policy classes. Moreover, the policy enforcement elements have been straightforwardly adapted to a wide range of computing environments. 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). This allows KAoS to be easily adapted to various agent computing environments (e.g., CORBA, Brahms, Cougaar, Web Services.) while maintaining a common code base and core functionality.

3. KAoS Policy Management in Cougaar

For the UltraLog project, KAoS is packaged as a set of Cougaar components, with extensions to allow use of the Cougaar message transport service and persistence mechanisms, as well as interfacing with Cougaar-specific policy enforcement mechanisms (binders).

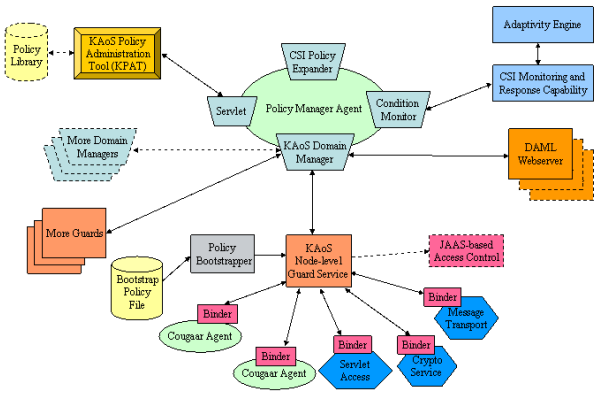


Figure 1. KAoS architecture in UltraLog

The KAoS architecture consists of the following components (figure 1):

- The KAoS Policy Administration Tool (KPAT) and associated servlet, providing a graphical user interface to domain and policy management functionality which is used to create and apply policies;
- The Policy Manager Agent, with the following plugins:
  - The KAoS Domain Manager (DM), which is responsible for registering agents consistent with policies on domain membership, performing policy conflict resolution, ensuring policy consistency at all levels of a domain hierarchy, notifying Guards in the event of a policy change, and storing policies in a secure repository. [1] Groups of people and computational entities are logically structured into 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. Membership in 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, membership in more than one domain at a time is possible. Domain membership may be defined extensionally (i.e., through explicit enumeration in a registry) or intentionally (i.e., by virtue of some common property such as a joint goal or a given place where various entities may be currently located);
  - The Policy Applicability Condition Monitor (aka Condition Monitor), which monitors external conditions (e.g. Cougaar OperatingModes) and proposes policies to the Domain Manager to be added or retracted consistent with pre-specified applicability conditions;
  - The Policy Expander, which expands high-level policies to a set of more finely grained policies before they are distributed to the Guard(s). Expansion is currently only useful for a small number of hand-crafted XML policies that are opaque to KAoS reasoning methods; OWL policies are expanded within the Domain Manager and sent directly to the Guards.
  - The KAoS Guard, which interprets policies that have been approved by the Domain Manager and assures enforcement with appropriate mechanisms, potentially including Cougaar binders, Java access control (JAAS), Nomads resource control, and obligation policy monitors and enablers.

[1] The DM currently delegates operational responsibility for some of these functions to the KAoS directory service, which contains Stanford's Java Theorem Prover (JTP; see below).

Integration with Cougaar

As mentioned previously, KAoS conforms to the Java Agent Services specification, which allows it to more easily interface with various agent architectures. For Cougaar, we developed a JAS-compliant message transport service layer which interacts with the Cougaar BlackboardService and uses Relays to send and receive messages. Furthermore, several KAoS components have been extended to take advantage of other features of the Cougaar architecture. The Guard is packaged as a Cougaar Service, and policy enforcers obtain a reference to it via the Cougaar ServiceBroker interfaces. Furthurmore, the Guard makes use of the Cougaar BlackboardService to provide persistence, storing the current policy set as well as other state information on the blackboard. Likewise, the KAoS Domain Manager is packaged as a Cougaar Plugin, and also uses the BlackboardService to provide persistence of policies, agent registrations, and other state information. Other KAoS components, such as the Condition Monitor and Policy Expander, were developed specifically for the UltraLog project and are implemented as Plugins.

Expressing Policies

The basic components of an authorization policy are the actor(s), modality (positive (permitted) or negative (forbidden)), action and properties relating to the action (e.g., target of the action, conditions). A sample policy would read as follows:

Actor(s) X is authorized to perform action(s) Y on target(s) Z.

An obligation policy also has two possible modalities: positive (required) or negative (not required). It also is associated with a trigger condition:

When actor(s) A performs action(s) B on target(s) C, then actor(s) X is obligated to perform action(s) Y on target(s) Z.

Actors, actions, and targets map to various classes and instances in the KAoS Policy Ontologies (KPO). For a given application, one may wish to write policies which refer to application-specific concepts. KAoS provides a flexible architecture which allows the introduction of application-specific concepts to the KPO. Concepts can be introduced by loading application-specific ontologies, or by registering concepts dynamically as they become known.

For example, consider the following two policies developed for the UltraLog program:

Members of CommunityX are authorized to communicate with Members of CommunityY using 3DES encryption.

A user in role PolicyAdministrator is authorized to access the servlet named PolicyManagementServlet.

In these examples, application-specific concepts such as communities, encryption levels, user roles, and servlet names have been introduced into the KPO. Note that these policies (as well as other UltraLog policies) are in the form of authorization policies, due to the nature of the enforcement capabilities developed for the program. An example of KAoS applications which use obligation policies can be found in [10].

In many cases, application-specific concepts may refer to dynamically changing data that does not lend itself well to introduction in the KAoS ontologies. In this case, the concept can be represented as a class in the ontology without defining membership. Policies refer to the concept's class, and class membership of individual instances is determined dynamically during policy disclosure queries through the use of Instance Classifiers. An Instance Classifier answers the question "Is instance x a member of class y?". Developers can write application-specific instance classifiers and register them with the Guard. The Guard then uses these classifiers when evaluating policy disclosure queries.

As an example, consider the communication policy above that refers to Cougaar communities. A community is a dynamically changing set of agents. Rather than constantly updating the definition of the community in the KAoS ontologies, the community is registered as a class (e.g. "CommunityX") without defining membership. A custom instance classifier for communities which interfaces with the Cougaar CommunityService is then registered with the Guard. When the Guard is called to evaluate a policy which refers to members of a community, it calls the instance classifier to ask "is the specified agent a member of this community", and uses the result to help determine whether the policy is applicable to the instance specified in the policy disclosure request.

Policy Conflict Detection

Both authorization and obligation policies also have another required component: priority. The priority of a policy is used to indicate its precedence in relation to other policies. Currently, priority is specified as an integer value; future work will allow complex logical expressions of policy precedence.

A policy is said to be directly in conflict with another policy if it is of the same priority, with an overlapping scope of actors, actions, and action context, but of opposing modality (figure 2).

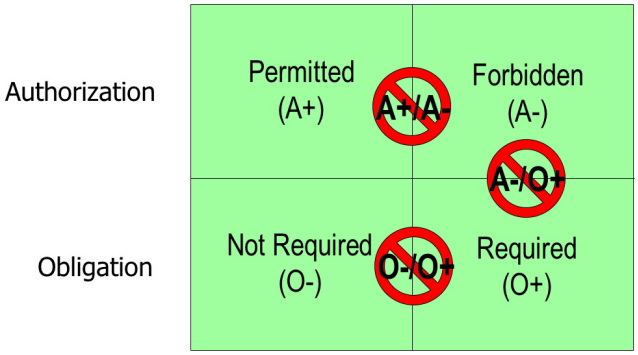


Figure 2. Three types of opposing modalities

Conflicting policies can be identified and, if desired, harmonized through the use of algorithms that we have implemented within Stanford's Java Theorem Prover (JTP; http://www.ksl.stanford.edu/software/JTP/). Investigation of means to help users discover various types of indirect policy conflicts is discussed in the Current Work section of this paper.

Policy Enforcement

Each KAoS Guard maintains a local repository of the current policies in force that are relevant to the actions or actors is it managing, as received from the Domain Manager. In this way, Guards can provide policy reasoning capabilities for policy enforcers, removing the burden of parsing and evaluating policies from the enforcer developer. The independence of the Guards from the Domain Manager, except at policy update time, allows enforcement of policies to continue locally even when communication with the DM may be temporarily unavailable. The representation of policy within Guards is in a simple look-up format that represents all the necessary semantics of the OWL policies yet is optimized for efficiency. The enforcer can perform various policy disclosure queries on the Guard to determine whether to allow an intercepted action to occur, asking questions such as "Is this action allowed?", "What are the allowed values for this property of a given action?", and "What are the obligations for the given action?". Thus the enforcer developer does not need to be concerned with the details of evaluating the applicability of policies, but can rather focus on the implementation of the enforcement capability itself.

Enforcers in Cougaar are in the form of Binders – infrastructure-level components which control access to infrastructure services. A number of enforcers (and instance classifiers) for security services have been developed with our collaborators at Cougaar Software, Inc. These enforcers interact with the KAoS Guard to check the authorization of intercepted actions. Enforcers have been developed for various kinds of message encryption and content filtering, servlet user access control and authentication, blackboard access control, and white pages (naming and lookup service) access control. The Binder implementations were developed by our Cougaar Software collaborators and the details of their implementation are thus beyond the scope of this paper.

Policy Templates

To assist non-specialists in defining sensible policies for a specific application, the KAoS Policy Administration Tool (KPAT) supports both a generic OWL policy editor, as well as policy templates tailored to specific applications. A policy template is a custom GUI used to create (and edit) a specific type of policy (or a set of policies) (figure 3). It hides certain properties and classes of the generic policy editor from users, presenting them with an application-specific subset of policy-relevant entities to choose from. Policy templates also reduce the burden of developers, allowing a single high-level policy to define multiple more-specific policies. For example, a policy represented in a template that forbids A communicating with B could be configured to generate four policies which represent the full range of protection intended by the policy developer: A being forbidden to send to B, B being forbidden to send to A, A being forbidden to receive from B, and B being forbidden to receive from A.

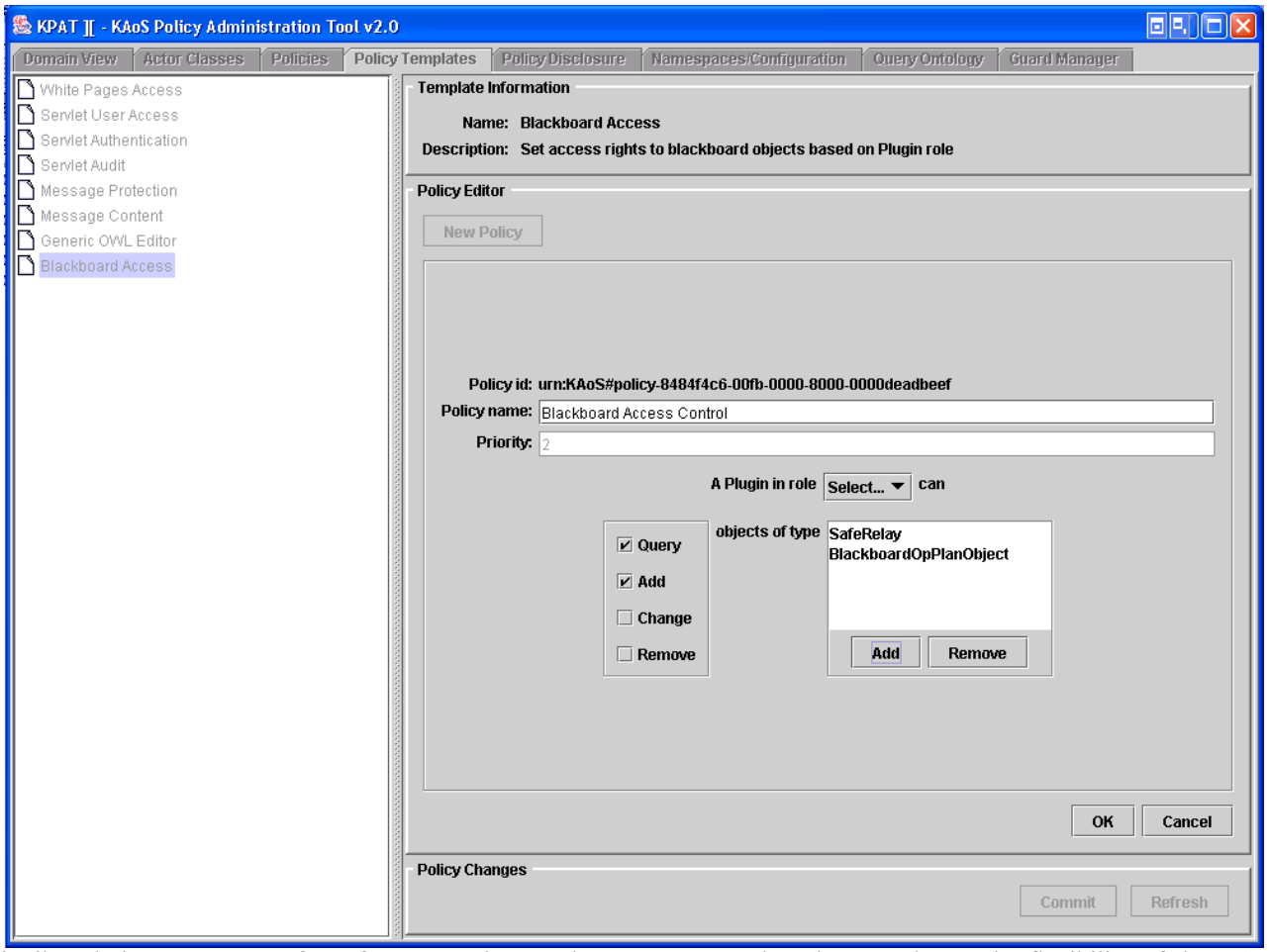
Several policy templates have been developed for the UltraLog program correlating with the specific enforcement capabilities described in the previous section. Currently, policy templates are implemented as custom-built Java classes. We expect to finish in the near future a graphical policy template editor that will allow a user to create new templates using the KPAT GUI.

Performance in Large Societies

A critical issue in the application of KAoS to the control of UltraLog security mechanisms was ensuring that policy disclosure queries from enforcers can be answered quickly enough as to not adversely affect the performance of the system. With this objective in mind, we optimized the policy disclosure methods such that response to a query is provided on average in less than 1ms[2]. Furthermore, queries can be executed concurrently by multiple enforcers, allowing KAoS to take advantage of multi-processor machines.

Performance of the KAoS Domain Manager is heavily dependent on the performance of the Java Theorem Prover (JTP) which provides the inference capabilities. Thus, assertions to the Domain Manager (e.g. policy updates, agent registrations, loading new ontologies, etc.) are bound by the performance of JTP. We have found that performance is acceptable even in large societies of over a thousand agents and hundreds of policies, where dynamic policy updates can be committed, deconflicted, and

[2] Tests were performed on a dual-processor Xeon machine with 2GB of RAM running Linux



distributed in a matter of a few seconds. Further We also plan to enhance the flexibility of the KAoS

Figure 3. Example KAoS policy template for UltraLog

enhancements to JTP (e.g., current work on general "untell" mechanisms) and advances in computer hardware will continue to improve this performance.

4. Current Work

In addition to the items already mentioned, current work includes policy administration scoping, distribution of knowledge among multiple policy managers, and user support for discovery of specific kinds of indirect policy conflicts.

Policy administration scoping will provide access control on domain management functionality, so that only authorized administrators will be able to view or modify entities or policies of a given class or within a given domain. Access control functionality will also be extended to policy disclosure queries, so that policy information is given only to those who have been authorized.

architecture to allow policy managers to share full or partial knowledge of policies and domain membership with other policy managers, consistent with what is permitted by knowledge disclosure policies. This will enhance the scalability of the policy management framework as well as providing redundancy in the policy repositories and reasoning capabilities to eliminate single point-of-failures.

Indirect policy conflict detection queries will be provided to help an administrator discover policy relationship information of interest. This will include, for example, queries to determine whether some policies are covered by other policies (even if they are not "in conflict"). Covered policies are those policies that have no effect because another policy covers the same (or broader) scope of actions but has greater (or equal) precedence. Another example would be in the case of queries to discover whether an agent can still perform its duties given the policies in effect. As an example of the latter, we want to avoid situations where basic

functionality of the system is not compromised due to overly restrictive policies put into force.

5. Conclusions

KAoS policy and domain management services have proven to be an effective and flexible solution to the dynamic control of security mechanisms in large-scale distributed systems. The flexible architecture of KAoS allows developers to extend the ontologies and plug in new enforcement capabilities without requiring changes to the policy management architecture itself, and without requiring the developer to have extensive knowledge of low-level reasoning and representation details. Policy disclosure queries have been optimized to execute quickly enough for real-time control of security mechanisms in a society of over a thousand medium-complexity agents and hundreds of policies. Moreover, the use of policy templates enables non-specialists to exercise policy-based control through the use of simple and easy-to-understand GUIs. Further enhancements will increase the power and convenience of these tools and capabilities.

6. Acknowledgements

This material is based on research sponsored by the Defense Advanced Research Projects Agency (DARPA) under agreement number F30602-00-2-0577. We appreciate the support of other IHMC KAoS researchers (Maggie Breedy, Larry Bunch, Paul Feltovich, Matt Johnson, Hyuckchul Jung, Shri Kulkarni, Niranjan Suri, William Taysom, Gianluca Tonti), as well as our sponsor at DARPA, Mark Greaves, and our collaborators at Cougaar Software (Richard Feiertag, Timothy Redmond, Sue Rho, and Sebastien Rosset). The U.S. Government and the Institute for Human and Machine Cognition are authorized to reproduce and distribute reprints and on-line copies of this paper and derivatives for their purposes notwithstanding any copyright annotation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of other parties.

7. References

[1] Bradshaw, J. M., Beautement, P., Raj, A., Johnson, M., Kulkarni, S., & Suri, N. (2004). 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. 355-400). Berlin: Springer Verlag.

[2] Bradshaw, J. M., Jung, H., Kulkarni, S., & Taysom, W. (2004). Dimensions of adjustable autonomy and mixed-initiative interaction. In M. Klusch, G. Weiss, & M. Rovatsos (Ed.), *Computational Autonomy*. (in press). Berlin, Germany: Springer-Verlag.

[3] Bradshaw, J. M., Uszok, A., Jeffers, R., Suri, N., Hayes, P., Burstein, M. H., Acquisti, A., Benyo, B., Breedy, M. R., Carvalho, M., Diller, D., Johnson, M., Kulkarni, S., Lott, J., Sierhuis, M., & Van Hoof, R. (2003). Representation and reasoning for DAML-based policy and domain services in KAoS and Nomads. *Proceedings of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2003)*. Melbourne, Australia, New York, NY: ACM Press,

[4] 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.

[5] 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,

[6] 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. In D. Fensel, K. Sycara, & J. Mylopoulos (Ed.), *The Semantic Web—ISWC 2003. Proceedings of the Second International Semantic Web Conference, Sanibel Island, Florida, USA, October 2003, LNCS 2870.* (pp. 419-437). Berlin: Springer.

[7] Uszok, A., Bradshaw, J. M., & Jeffers, R. (2004). KAoS: A policy and domain services framework for grid computing and semantic web services. *Proceedings of the Second International Conference on Trust Management*. Oxford, England,

[8] Uszok, A., Bradshaw, J. M., Jeffers, R., Johnson, M., Tate, A., Dalton, J., & Aitken, S. (2004). Policy and contract management for semantic web services. *AAAI 2004 Spring Symposium Workshop on Knowledge Representation and Ontology for Autonomous Systems*. Stanford University, CA, AAAI Press. To appear in IEEE Intelligent Systems.

[9] Uszok, A., Bradshaw, J. M., Jeffers, R., Suri, N., Hayes, P., Breedy, M. R., Bunch, L., Johnson, M., Kulkarni, S., & Lott, J. (2003). KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. *Proceedings of Policy 2003*. Como, Italy,