The KAoS Policy Services Framework

Jeffrey M. Bradshaw, Andrzej Uszok, Maggie Breedy, Larry Bunch, Thomas C. Eskridge, Paul J. Feltovich, Matthew Johnson, James Lott, and Micael Vignati

Florida Institute for Human and Machine Cognition (IHMC), Pensacola, FL
{jbradshaw, auszok, mbreedy, lbunch, teskridge, pfeltovich, mjohnson, jlott, mvignati}@ihmc.us

ABSTRACT

The KAoS policy management framework pioneered the use of semantically rich ontological representation and reasoning to specify, analyze, deconflict, disseminate, enforce, and monitor the operation of digital policies. The framework has continued to develop over more than a decade, inspired by both technological advances and the practical needs of its varied applications.

1. INTRODUCTION

In order to cater to continuously changing conditions and resources, complex distributed systems must be capable of adapting their behavior while they are still running. Digital policy provides a means to dynamically regulate the behavior of systems at runtime without requiring the consent or cooperation of the components being governed.

The flexibility and power of a policy management framework is to a large degree determined by the expressivity and computational efficiency of its policy representation. The KAoS Policy Services framework [4; 5] was the first to offer an ontology-based approach and is currently the most successful and mature of such efforts. Although competing approaches for policy representation have been advanced, the endorsement by the United States National Security Agency's (NSA) Digital Policy Management (DPM) standards effort of the use of ontologies to bridge the gap between natural language expressions and machine-interpretable specifications has highlighted the value of an ontology-based approach [2]. Following collaborative efforts by the DPM Architecture Group and IHMC, the KAoS core ontology was adopted as a common basis for future ontology-based standards efforts [6]. Selected DPM challenges and corresponding rationale for an ontology-based policy approach are listed in Table 1 below. For a more detailed comparison and assessment of different approaches to policy representation and reasoning, see [1; 3].

Policy generated at multiple levels of an Enterprise cannot be shared	Can represent policy from multiple perspectives and at multiple levels of abstraction
Difficult to identify conflicting or inconsistent policies	Efficient description-logic-based deconfliction algorithms
Multiple policy languages	Expressiveness of OWL semantics obviates the need for multiple special-purpose languages
Current implementations not integraged across Enterprise	Can provide end-to-end system integration across multiple policy domains
Difficulty in translating policies from one language to another	Expressiveness of OWL semantics enables automatic translation of special-purpose policy languages when necessary

Table 1. How OWL-based Policy Addresses DPM Challenges.

2. KAOS OVERVIEW

We have learned that a general purpose ontology-based policy framework such as KAoS needs to make the sophistication, flexibility, and power of ontological representation and reasoning available to people in a simple and understandable manner. In response to this requirement, we have developed a three-layer policy management architecture that ensures consistency among these separate but interdependent sets of components.

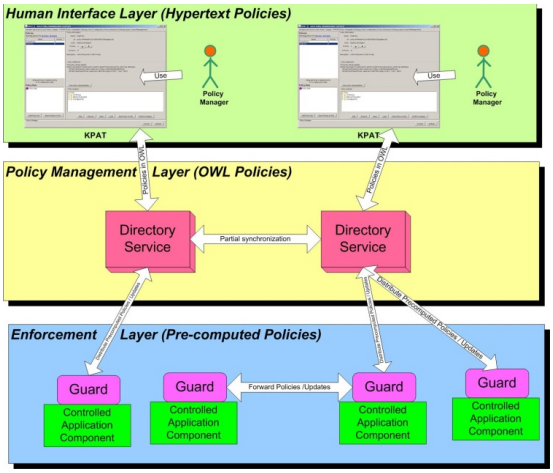

Figure 1. KAoS Conceptual Architecture**.**

2.1 Three-Layered Architecture

The basic elements of the KAoS architecture are shown in Figure 1. The three layers of functionality correspond to three different policy representations:

- *Human Interface Layer*: This layer uses a hypertext-like graphical interface called KPAT (KAoS Policy Administration Tool — pronounced as KAY-pat) for policy specification in the form of constrained English sentences. The vocabulary is automatically provided from the relevant core ontologies or application-specific ones. Besides KPAT's use in policy specification and analysis, it is employed for administrative tasks such as browsing and loading ontologies, and domain and Guard management. The generic KPAT interface can be easily customized using policy templates or replaced entirely with a custom user interface.
- *Policy Management Layer*: Within this layer, OWL is used to encode and manage policy-related information. The KAoS Distributed Directory Service (DDS) encapsulates a set of ontology reasoning mechanisms used for policy deconfliction, analysis, and testing.
- *Policy Monitoring and Enforcement Layer*: KAoS automatically "compiles" OWL policies to a very efficient format that can be used for monitoring and enforcement. This representation provides the grounding for abstract ontology terms, connecting them to the instances in the runtime environment and to other policy-related information (e.g., dynamic state or history). Extensibility is supported through a framework with well-defined interfaces that can be enriched to support new kinds of policies. KAoS Guards residing in this layer of the software architecture are integrated with the controlled application and provide an API for policy information querying and decision-making.

Within each of the layers, the third-parties may plug-in specialized extension components if needed, as described in more detail below. Such components are typically developed as Java classes and described using ontology concepts in the configuration file. They can then be used by KAoS in policy specification, reasoning and enforcement.

2.2 Use of Ontologies in KAoS

Core and application-specific ontologies. The KAoS core policy ontology consists of a set of independent OWL files (available at: http://ontology.ihmc.us/ontology.html). They define the root concepts for policy-governed actions, actors, places, states, history, and situations. Application developers normally extend the core ontology with additional application-specific classes, properties, and individuals that can be used as vocabulary in policy definitions. It is also possible to link application-defined concepts to any number of pre-existing ontologies.

Grounding ontologies in the world. Ontology-based policy services dynamically define mappings between class definitions and entities in the controlled environment and in the world. This can be accomplished in different ways. First, static elements of the environment may be defined as individuals in application-specific ontologies. For instance, we might describe specific instances of robots, a range of existing radio spectrums, or a set of producers of weather reports for a given area. In addition, dynamic elements of the environment may be registered within KAoS through the Guard interface, or information about them can be provided to a Guard at policy enforcement time. For instance, at runtime, new areas of robot operation can be defined, new weather reports can be produced, new radios can be introduced, and new domains, roles, or teams can be formed or removed.

2.3 KAoS Policy Semantics

Two main types of KAoS policies. KAoS supports two main types of policy: authorization and obligation. The set of permitted actions is determined by authorization policies that specify which actions an actor or set of actors is allowed (*positive authorization policies*) or not allowed *(negative authorization policies*) to perform in a given context. Obligation policies specify actions that an actor or set of actors is required to perform (*positive obligations*) or for which such a requirement is waived (*negative obligations*). All other kinds of policies (e.g., delegation, teamwork coordination) are built from these two primitive types.

Basic form of KAoS policies. The basic form of KAoS policies is as follows:

[Actor] is [constrained] to perform [controlled action] which has [any attributes]

[Actor] is a variable that refers to the subject of the policy-controlled action. Any of the following can be defined as actors:

- A single actor instance (e.g. Robot32);
- An actor class or a role (as in role-based access control) using an actor class name (e.g. members of class Robot, Weather Producer, Team A; all Robots within 50 feet of my current location);
- The complement of some instance or set of instances (e.g. any Robot except Robot324);
- The complement of actor class or set of classes (e.g. any Robot that is not Pioneer).

[constrained] is a variable that refers to the basic type of the policy (i.e., positive or negative authorization, positive or negative obligation).

[controlled action] is a variable that refers to the action class that will be controlled by the policy (e.g. Radio Transmission, Movement).

[any attributes] is an optional variable referring to one or more attributes of the controlled action. For example, a Radio Transmission action may have attributes defining configuration parameters, the destination of the transmission, and so forth. These attributes will typically be used to describe aspects of context relating to the controlled action.

Attributes can be used either as part of simple value restrictions or to define a test that dynamically relates two or more separate attributes. A simple restriction typically has the form:

[all | some] [attribute] values are [within the set of enumerated instances | of a given type]

For example, such a restriction allows a policy to say that:

- A valid credential for a given user must be one of the set of recognized credentials.
- Receivers of a given radio transmission must all be holders of a particular security clearance.

Some policies require the definition of dynamic attributes whose values must be tested relative to the values of some other attribute. Support in KAoS for this feature allows users to define policies that relate to the local context of the action or actor. For example:

- A robot is authorized to request assistance only from current members of its team.
- Employees are forbidden from using printers belonging to departments other than their own.
- Users are authorized to share documents only if they share a common credential.

Special-purpose reasoning. OWL semantics do not allow the expression of the constraints on attributes described above. The KAoS role-value-map reasoner solves this problem. In addition, KAoS implements a special-purpose reasoner for spatial relations. KAoS uses the OWL-Time ontology to reason about time. We have developed special-purpose mechanisms to handle delegation of authority. We have also developed a meta-reasoner named Kaa that uses probabilistic information and maximization of expected value to make decisions about policy exceptions (e.g., exceptional relaxation of policy constraints) or to incorporate complex uncertain runtime information (e.g., risk-adaptive access control). KAoS policy refinement mechanisms map from high-level policies (e.g., expressions of "Commander's intent," or mission-level objectives) to low-level policies that dictate operational aspects of network configuration and operation. As our research progresses, we will upgrade our initial static mapping approaches with more advanced synthetic methods supported by a planner.

Obligation policy triggers. Unlike authorization policies, obligation policies include triggers that specify the conditions under which the required action will be activated (e.g., When two hours have elapsed, the operator must terminate the transmission). Trigger actions are specified in a manner that is similar to controlled actions. *Relative attributes* are typically used to relate the trigger to the obliged action (e.g., "If [some robot] fails, then [some robot] must notify its teammates; "If [some message] is of class 'secret' or greater, [some message] must be logged to the audit queue)."

History and current state of a situation. All policies are defined in the context of a *Situation*, which possesses a history and a set of variables describing its current state. History is used to qualify the

applicability of the policy relative to past events, while state information is used to qualify the applicability of a policy relative to values representing the current state of one or more variables.

Policy precedence reasoning. The ranking of policies by order of importance is used in two important phases of policy management. In the first case, when a policy is created or updated, the policy service must determine whether the new policy is consistent with the existing set of policies. If the new policy and an existing policy have the same ranking in importance, cover overlapping actor, action, and context classes, and have conflicting modalities (i.e., authorized/forbidden, required/forbidden, required/not required), the new policy is rejected and deconfliction recommendations are given to the user. If the new policy overlaps with an existing policy and has a conflicting modality, but one of the two policies has a higher ranking than the other, no deconfliction is required.

The second case occurs during authorization policy decisions (i.e., determining whether or not an action is permitted). As part of this process, KAoS collects a set of policies with action classes positively classifying the action instance being tested. Policy ranking allows KAoS to group applicable policies into sets of decreasing importance. At policy creation time, the consistency checking mechanism has already assured that the sets are not in conflict. At policy decision time, within the policy set with the highest priority for a given action, a single positive or negative authorization policy will determine whether the action is permitted or forbidden.

KAoS originally relied exclusively on numeric policy priority assignments by users to determine how policies should be ranked. This mechanism has important performance advantages. However, a disadvantage of this approach has been that people may have difficulty assigning meaningful priorities and tracking how a given policy's priority relates to the priorities of other policies, especially when integrating large numbers of policies from different sources. For this reason, we have extended the priority mechanism in KAoS to use a powerful logical precedence mechanism in addition to numeric priorities. This allows administrators to specify an almost-infinite variety of precedence relationships among policies (e.g., Jim Hanna's policies take precedence over anyone else's policies; policies of the domain administrator (a role) take precedence over user (another role) policies; more recent policies take precedence over older policies; superdomain policies take precedence over subdomain policies; policies for Pioneer robots take precedence over policies for the general robot class; policies about writing to a specific directory take precedence over policies about writing to the volume; negative authorizations take precedence over positive authorizations).

3. HUMAN INTERFACE LAYER

KPAT's generic Policy Editor presents an administrator with a starting point for policy construction — essentially, a very generic policy statement shown as hypertext. Clicking on a specific link in this statement that represents a variable provides users with menu choices allowing them to make the generic policy more specific.

Policies defined using this menu-driven process follow a predetermined syntax in constrained natural language for either authorization or obligation policies. Authorization policies permit or forbid some action while obligation policies either require some action to be performed or waive such a requirement. Figure 2 shows an example of an authorization policy being defined in KPAT.

During use, KPAT accesses the ontologies that have been loaded into the DDS and provides the user with the list of choices narrowed to the current context of the policy construction. New ontology classes and instances needed for specific kinds of policies can also be created within KPAT. Since the ontologies directly determine what choices are provided to users when they build policies,, the correctness of the semantics of the policy is dependent only on the correctness of the ontology. KAoS tools to automate the process of creating ontologies from the environment (e.g., SNMP, WSDL, Java) are designed to further ensure the correctness of the ontology. The translation from the form of the constrained English policy to its OWL representation in KPAT is deterministic, assuring both reliability and consistency.

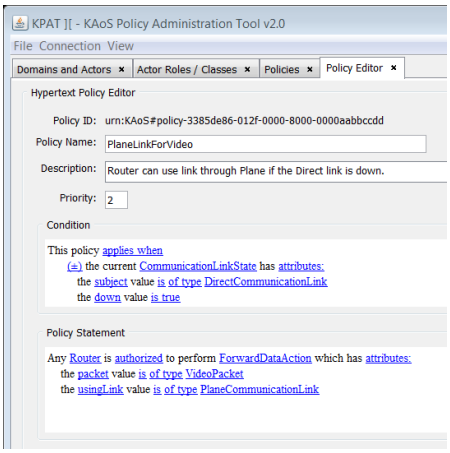

Figure 2. Authorization Policy in the KPAT Generic Policy Editor.

To further simplify policy construction, KPAT provides a *Policy Template Editor* that allows custom policy editors for a given kind of policies to be created by point-and-click methods. For instance, if an application will require the definition of several policies governing publish/subscribe actions, a custom policy editor can be quickly created by limiting choices to just what is needed, thus eliminating the requirement for repetitive selections when a given type of policy has to be created multiple times.

As another example of KPAT extensibility, when filling in values of type "area," users are presented with a custom area editor. The editor allows them to define a polygonal region on top of a domain-specific background map by using the mouse to define edge points.

4. POLICY MANAGEMENT LAYER

This layer mediates between the human interface layer and the monitoring and enforcement layer. Though other kinds of reasoning take place in the top and bottom layers, the middle layer is where virtually all the ontological reasoning and representation takes place. The higher computational cost of reasoning for policy deconfliction and analysis is paid upfront so that policy monitoring and enforcement in the lowest layer can be performed in a highly efficient manner.

4.1 Bootstrapping and Basic Policy Reasoning

An entire KAoS configuration, including application-specific ontologies and policies, can be captured declaratively as OWL and reused at a later time. During bootstrap, the core policy ontology (section 2.2) is loaded into the ontology reasoners integrated with KAoS. After bootstrap, application-specific ontologies may be loaded. The reasoner maintains information about domain structures, registered actors and other entities

pertinent to the situation. With respect to policy management, the reasoner supports the creation of policies by supplying KPAT with lists of vocabulary terms (e.g., all of the action classes which can be performed by a given class of actor). Policies can also be created, of course, through a programmatic interface. During policy analysis the reasoner finds relationships among action classes controlled by policies. As policies are distributed to Guards (see below), the reasoner classifies existing instances (e.g., the list of actors) so that relevant information of other kinds can be sent to the Guards at the same time.

Policy dissemination. When policies are added and modified, or when a Guard connects or reconnects to the Distributed Directory Service (DDS), the DDS assembles the appropriate update information relevant to a particular Guard. An alternative mechanism allows direct communication among a group of Guards for the exchange of policy and cache updates. We have designed this capability for a mobile ad-hoc network environment where continuous direct communication between the DDS and the Guards is not always possible. When networked communication is not available (e.g., disconnected sensors), policies can be directly loaded into a standalone Guard.

During the policy dissemination process, the OWL policy representations are converted by the DDS to a form that enables the Guards to make complex enforcement decisions very efficiently. The algorithm traverses the OWL policy structure and "compiles" it into a hash-table-like structure within the Guards.

5. ENFORCEMENT LAYER

The Guard is where KAoS meets the controlled system. Its primary role is as a policy decision point. Guards register to receive policies about particular entities or classes of entities for a given set of action classes. Because Guards can save their policies and reload them directly from a snapshot, they can be bootstrapped in a standalone mode without a need to connect to the DDS. This functionality allows policies to govern the actions of standalone sensors or similar components.

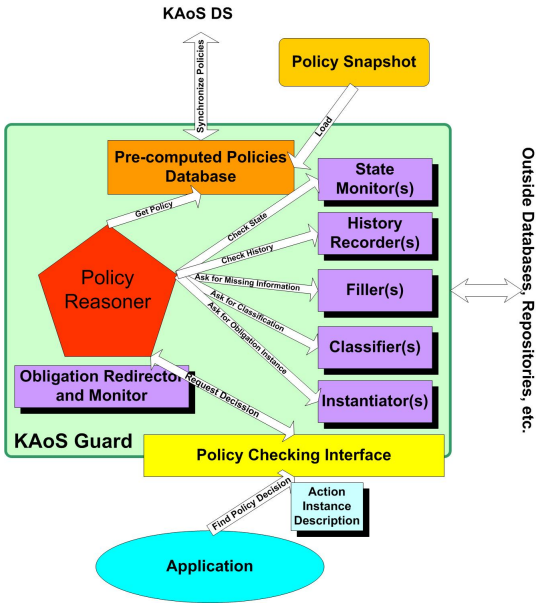

Figure 3. Conceptual Architecture of the KAoS Guard.

Guards not only receive information about policies, but also about the state of the system and the entities being managed. Guards do not by themselves provide monitoring functions, but they do provide interfaces to plug in outside monitors or databases providing access to external state or event-related information.

The KAoS Guard Policy Checking Interface provides a set of methods that allow checking for:

- *Authorization.* If an action is not authorized, an exception is thrown with information about the policy that prevented it. In some secure applications, however, it would not be desirable to release information about the cause of the policy exception, so we allow this to be controlled by policy.
- *Obligations.* A list of obligations for a given actor is returned, sorted in rank order of importance. In addition, if there are obligations for other actors that are triggered by an external event, then KAoS will try to locate them and forward the obligations to them.
- *Configuration options.* If a partial description of the action is sent to KAoS, a range of allowed values for properties of a given action is returned. For instance, if an application were to query the Guard about a planned radio transmission, information about the maximum power and range of frequencies allowed to be used in the given geographical area would be returned to it. Disclosure policies would be used to filter out unauthorized information in the results.

In order to support the semantics of complex application-specific policies, Guards accommodate a variety of extensions. These can be activated on demand, as specified in each Guard's configuration information. Specific extensions are:

- *History Monitor:* tracks the history of specific actions and allows verifying whether a given history is present (e.g., three successive login failures; obligation fulfillment).
- *State Manager:* manages set of environment-specific sensors that provide information about dynamic aspects of the environment or situation (e.g., threat level, resource availability, locations in time or space).

We are interested in hosting the KAoS Guard on a tamper-proof platform (e.g., the DRS Secure Core Module) for deployment of KAoS in sensitive environments.

We believe that the KAoS ontology-based policy management approach holds great promise for the challenges of complex distributed applications. Due to space limitations, we have been able to describe KAoS only in limited fashion. Readers are referred to [4; 5] and to http://ontology.ihmc.us/ontology.html for more complete information.

6. REFERENCES

1. Bradshaw, J.M., A. Uszok, and R. Montanari. "Policy-Based Governance of Complex Distributed Systems: What Past Trends Can Teach Us about Future Requirements." In *Agile Computing*, edited by N. Suri. in preparation.
2. National Security Agency (NSA), Enterprise Services D., Identity and Access Management Branch 2012. Digital policy management: A foundation for tomorrow. In *RSA Conference*. https://ae.rsaconference.com/US12/scheduler/eventcatalog/eventCatalog.do. (accessed November 29, 2012).
3. Tonti, G., J.M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. "Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder." In *The Semantic Web—ISWC 2003. Proceedings of the Second International Semantic Web Conference, Sanibel Island, Florida, USA, October 2003, LNCS 2870*, edited by D. Fensel, K. Sycara, and J. Mylopoulos, 419-437. Berlin: Springer, 2003.
4. Uszok, A., J.M. Bradshaw, M.R. Breedy, L. Bunch, P. Feltovich, M. Johnson, and H. Jung. "New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of KAoS." In *Proceedings of the 2008 IEEE Conference on Policy*. Palisades, NY, 2008.
5. Uszok, A., J.M. Bradshaw, J. Lott, M. Johnson, M. Breedy, M. Vignati, K. Whittaker, K. Jakubowski, and J. Bowcock. "Toward a Flexible Ontology-Based Policy Approach for Network Operations Using the KAoS Framework." Presented at the The 2011 Military Communications Conference (MILCOM 2011) 2011, 1108-1114.
6. Westerinen, A. "Digital Policy Management Ontology Discussion." Presented at the NSA Digital Policy Management Technical Exchange Meeting, Washington, DC, January 25, 2012.