An Ontology-Based Representation for Policy-Governed Adjustable Autonomy

Hyuckchul Jung, Jeffrey M. Bradshaw, Shri Kulkarni, Maggie Breedy, Larry Bunch, Paul Feltovich, Renia Jeffers, Matt Johnson, James Lott, Niranjan Suri, William Taysom, Gianluca Tonti, & Andrzej Uszok

Institute for Human and Machine Cognition (IHMC), 40 S. Alcaniz, Pensacola, FL 32502
{jbradshaw, hjung, skulkarni, mbreedy, lbunch, pfeltovich, rjeffers, mjohnson, jlott, nsuri, wtaysom, gtonti, auszok}@ihmc.us

Abstract

Policies are a means to dynamically regulate the behavior of system components without changing code nor requiring the cooperation of the components being governed. By changing policies, a system can be continuously adjusted to accommodate variations in externally imposed constraints and environmental conditions. KAoS policy and domain services rely on an OWL ontology of the computational environment, application context, and the policies themselves that enables runtime extensibility and adaptability of the system, as well as the ability to analyze policies relating to entities described at different levels of abstraction. Besides the currently implemented conflict detection and resolution methods, we are developing an approach to determine how and when to make policy changes based on adjustable autonomy considerations. This approach relies heavily on the information contained in the KAoS Policy Ontologies.

Introduction

As computational systems with increasing autonomy interact with humans in more complex ways-and with the welfare of the humans sometimes dependent on the conduct of the agents-there is a natural concern that the agents act in predictable ways so that they will be acceptable to people [1]. In addition to traditional concerns for safety and robustness in such systems, there are important social aspects relating to predictability, feedback, order, and naturalness of the interaction that must be attended to [3]. This paper summarizes our efforts to address some of the technical and social aspects of agent design for increased human acceptability through an ontology-based representation of policy in autonomous systems. From a technical perspective, we want to ensure the protection of agent state, the viability of agent communities, and the reliability of the resources on which they depend. To accomplish this, we must guarantee insofar as possible that the autonomy of agents can always be bounded by explicit enforceable policy that can be continually adjusted to maximize their effectiveness and safety in both human and computational environments.

From a social perspective, we want agents to be designed so as to fit well with how people actually work together. Explicit policies governing human-agent interaction based on careful observation of work practice and an understanding of current social science research can help assure that effective and natural coordination, appropriate levels and modalities of feedback, and adequate predictability and responsiveness to human control are maintained. These factors are key to providing the reassurance and trust that are prerequisite to the widespread acceptance of autonomous agent technology for non-trivial applications.

Policy and Autonomy

Policies are a means to dynamically regulate the autonomy of system components without changing code nor requiring the cooperation of the components being governed. By changing policies, a system can be continuously adjusted to accommodate variations in externally imposed constraints and environmental conditions. Policies are becoming an increasingly popular approach to dynamic adjustability of applications in academia and industry (http://www.policy-workshop.org/). Elsewhere we have pointed out the many benefits of policy-based approaches, including reusability, efficiency, extensibility, context-sensitivity, verifiability, support for both simple and sophisticated components, protection from poorly-designed, buggy, or malicious components, and reasoning about component behavior [1]. Policies have important analogues in animal societies and human cultures [8].

Dimensions of autonomy. Some important dimensions relating to adjustable autonomy and mixed initiative interaction can be straightforwardly characterized by reference to figure 1.[1] Note that there are two basic dimensions:

[1] See [2] for a more complete discussion of these dimensions. We can make a rough comparison between some of these dimensions and the aspects of autonomy described by Falcone and Castelfranchi [7]. Environmental autonomy can be expressed in terms of the possible actions available to the agent—the more the behavior is wholly

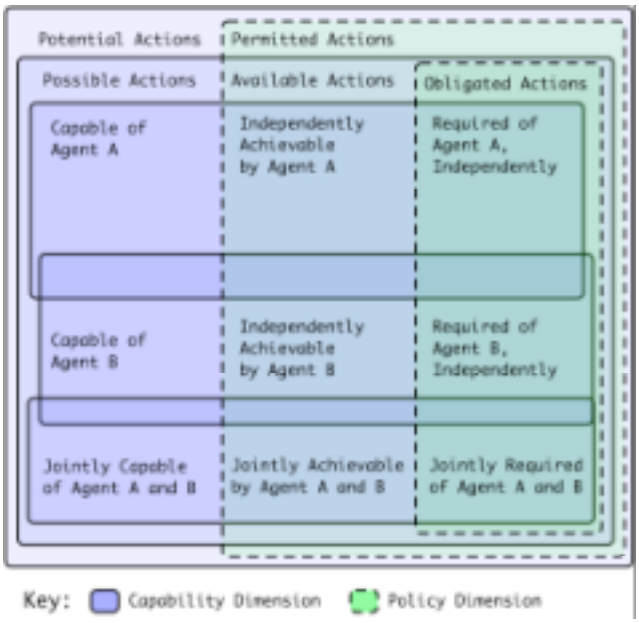

Figure 1. Basic dimensions of adjustable autonomy and mixed-initiative interaction.

- a *descriptive* dimension corresponding to the sense of autonomy as self-sufficiency that stretches horizontally to describe the actions an actor in a given context is *capable* of performing; and
- a *prescriptive* dimension corresponding to the sense of autonomy as self-directedness running vertically to describe the actions an actor in a given context is allowed to perform or which it must perform by virtue of *policy* constraints in force.

The outermost rectangle, labeled *potential actions,* represents the set of all actions defined in some ontology under current consideration.[1] In other words, it contains the union of all actions for all actors currently known to the computational entities that are performing reasoning about adjustable autonomy and mixed-initiative interaction. Note that there is no requirement that all actions that an agent may take be represented in the ontology; only those which are of consequence for policy representation and reasoning need be included.

The rectangle labeled *possible actions* represents the set of potential actions whose achievement by some agent is deemed sufficiently imaginable in the current context. Of these possible actions, any given actor[2] (e.g., Agent A) will likely only be deemed to be *capable of* performing some subset. Capability is a function of the *abilities* and *resources* available to an actor attempting to undertake some action. An actor's ability is the sum of its own knowledge and skills, whereas its resources consist of all other assets it can currently draw on in the performance of the action. Two actors, Agent A and Agent B, may have both overlapping and unique capabilities.[3] If a set of actors is *jointly capable of* performing some action, it means that it is deemed to be possible for it to be performed by relying on the capabilities of both actors. Some actors may be capable of performing a given action either individually or jointly; other actors may not be so capable.

Along the prescriptive dimension, policies specify the various *permissions* and *obligations* of actors [6]. *Authorities* may impose or remove involuntary policy constraints on the actions of actors. Alternatively, actors may voluntarily enter into *agreements* that mutually bind them to some set of policies so long as the agreement is in effect. The *effectivity* of an individual policy is the set of conditions that determine when it is in or out of force.

The set of *permitted actions* is defined by *authorization policies* that specify which actions an actor is allowed (*positive authorizations* or *A+* policies) or not allowed (*negative authorizations* or *A-* policies) to perform in a given context. The intersection of what is possible and what is permitted to a given set of actors defines a set of *available actions.*

Of those actions that are available to a given actor, some subset may be judged to be *independently achievable* by it in the current context. Some actions, on the other hand, would only be *jointly achievable.*

Finally, the set of *obligated actions* is defined by *obligation policies* that specify actions that an actor is required to perform (*positive obligations* or *O+* policies) or for which such a requirement is waived (*negative obligations* or *O-* policies). Positive obligations commit the resources of actors, reducing their current overall capability accordingly. *Jointly obligated actions* are those that two or more agents are explicitly required to perform.

A major challenge is to ensure that the degree of autonomy is continuously and transparently adjusted to be consistent with explicitly declared policies which themselves can, ideally, be imposed and removed at any time as appropriate [11]. For example, one goal of the agent or external entity performing such adjustments should be to make sure that the range of permissible actions do not exceed the range of those that are likely to be achievable by the agent.[4] While

deterministic in the presence of a fixed set of environmental inputs, the smaller the range of possible actions available to the agent. The aspect of self-sufficiency in social autonomy relates to the ranges of what can be achieved independently vs. in concert with others; deontic autonomy corresponds to the range of permissions and obligations that govern the agent's choice among actions.

[1] The term *ontology* is borrowed from the philosophical literature, where it describes a theory of what exists. Such an account would typically include terms and definitions only for the very basic and necessary categories of existence. However, the common usage of ontology in the knowledge representation community is as a vocabulary of representational terms and their definitions at any level of generality. A computational system's "ontology" defines what exists for the program—in other words, what can be represented by it.

[2] For discussion purposes, we use the term *actor* to refer to either a biological entity (e.g., human, animal) or an artificial agent (e.g., software agent, robotic agent).

[3] Note that although we show A and B sharing the same set of possible actions in figure 1, this is not necessarily the case.

[4] If the range of achievable actions for an agent is found to be too restricted, it can, in principle, be increased in any combination of four ways: 1. removal of some portion of the environmental constraints, thus increasing the range of possible actions; 2. increasing its permissions; 3. making additional external help available to the agent, thus increasing its

the agent is constrained to operate within whatever deontic bounds on autonomy are currently enforced as authorization and obligation policies, it is otherwise free to act.

Adjustable autonomy. A major challenge in the design of intelligent systems is to ensure that the degree of autonomy is continuously and transparently adjusted through mechanisms such as policy so as to meet whatever performance expectations have been imposed by the system designer and the humans and agents with which the system interacts. We note that is not the case that "more" autonomy is always better:[1] as with a child left unsupervised in city streets during rush hour, an unsophisticated actor insufficiently monitored and recklessly endowed with unbounded freedom may pose a danger both to others and itself. On the other hand, a capable actor shackled with too many constraints will never realize its full potential.

Thus, a primary purpose of adjustable autonomy is to maintain the system being governed at a sweet spot between convenience (i.e., being able to delegate every bit of an actor's work to the system) and comfort (i.e., the desire to not delegate to the system what it can't be trusted to perform adequately).[2] Assurance that agents will operate safely within well-defined bounds and that they will respond in a timely manner to external control is required for them to be acceptable to people in the performance of non-trivial tasks. People need to feel that agents will handle unexpected circumstances requiring adjustment of their current state of autonomy flexibly and reliably. To the degree adjustable autonomy can be successfully implemented, agents are kept, to the degree possible, from exceeding the limits on autonomy currently in effect, while being otherwise free to act in complete autonomy within those limits. Thus, the coupling of autonomy with adequate autonomy adjustment mechanisms gives the agent maximum opportunity for local adaptation to unforeseen problems and opportunities while assuring humans that agent behavior will be kept within desired bounds.

All this, of course, only complicates the agent designer's task, a fact that has lent urgency and impetus to efforts to develop broad theories and general-purpose frameworks for adjustable autonomy that can be reused across as many agents, domains, and applications as possible. To the degree that adjustable autonomy services can be competently implemented and packaged for convenient use within popular development platforms, agent designers can focus their attention more completely on the unique capabilities of the individual agents they are developing while relying on the extant services to assist with addressing cross-cutting human-agent interaction concerns.

joint capabilities; or 4. reducing an agent's current set of obligations, thus freeing resources for other tasks. Of course, there is a cost in computational complexity to increasing the range of actions that must be considered by an agent—hence the judicious use of policy where certain actions can either be precluded from consideration or obligated with confidence in advance by a third party.

[1] In fact, the multidimensional nature of autonomy argues against even the effort of mapping the concept of "more" and "less" to a single continuum.

[2] We note that reluctance to delegate can also be due to other reasons. For example, some kinds of work may be enjoyable to people—such as skilled drivers who may prefer a manual to an automatic transmission.

KAoS Policy and Domain Management Services

KAoS a collection of componentized policy and domain management services compatible with several popular agent frameworks, including Nomads [13], the DARPA CoABS Grid [10], the DARPA ALP/Ultra*Log Cougaar framework (http://www.cougaar.net), CORBA (http://www.omg.org), Voyager (http://www.recursionsw.com/osi.asp), and Brahms (www.agentisolutions.com). While initially oriented to the dynamic and complex requirements of software agent applications, KAoS services are also being adapted to general-purpose grid computing (http://www.gridforum.org) and Web Services (http://www.w3.org/2002/ws/) environments as well [9; 17]. KAoS has been deployed in a wide variety of applications, from coalition warfare [4; 14] and agile sensor feeds [15], to process monitoring and notification [5], to robustness and survivability for distributed systems [http://www.ultralog.net], to semantic web services composition [17], to human-agent teamwork in space applications [3], to cognitive prostheses for augmented cognition [1]. See [1] for a more complete description of policy issues and a summary of these and other applications.

KAoS domain services provide the capability for groups of software components, people, resources, and other entities to be organized into domains and subdomains to facilitate agent-agent collaboration and external policy administration.

KAoS policy services allow for the specification, management, conflict resolution, and enforcement of policies within domains.

KAoS Policy Ontologies. The current version of the core KAoS Ontologies (*http://ontology.ihmc.us/)* defines basic concepts for actions, actors, groups, places, various entities related to actions (e.g., computing resources), and policies. It includes more than 100 classes and 60 properties.

The core *actor ontology* contains classes of people and software components that can be the subject of policy. 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 intentionally (i.e., by virtue of some common property such as types of credentials actors possess, or a given place where various entities may be currently located).

The core *action ontology* defines various types of basic actions such as accessing, communication, monitoring, moving, and so forth. An ontological definition of an action associates with it a list of properties describing context of this action or a current state of the system relevant to this action. Example properties of action classes are, for instance: destination of the communication, type of

encryption used, resources accessed, time, previous history, and so forth. Each property is associated with the definition of a range of values it could have for each of the action classes. A particular instance of the action class can take values on the given property only from within this range. Actions are also divided into *ordinary actions* and *policy actions,* the latter comprising those actions that have to do with the operations of the KAoS services themselves[1].

For a given application, the core KAoS ontologies are usually further extended with additional classes, individuals, and rules, which use the concepts defined in the core ontologies as superconcepts. This allows the framework to discover specialized concepts by querying an ontology repository for subclasses or subproperties of the given concept or property from the core ontologies. For example additional application-related context could be added to actions such as specific credentials used in a given environment.

During the initialization process, the core policy ontologies are loaded into the *KAoS Directory Service* using the namespace management capabilities of the *KAoS Policy Administration Tool (KPAT)* graphical user interface. Additional application-specific or platform-specific ontologies can then be loaded dynamically using KPAT or programmatically using the appropriate Java method. A distributed version of the KAoS Directory Service is currently being implemented. We are also studying possibilities for interaction among multiple instances of Policy Services [17].

The Directory Service is also informed about the structure of policies, domains, actors, and other application entities. This information is added to the ontology repository as instances of concepts defined in pre-loaded ontologies or values of these instance properties. As the end-user application executes, instances relating to application entities are added and deleted as appropriate.

KAoS employs the Jena Semantic Web Toolkit by HP Labs in Bristol (*http://www.hpl.hp.com/semweb*) to incrementally build OWL definitions and to assert them into the ontology repository managed by the Directory Service. In order to provide description logic reasoning on the OWL defined ontologies, the Java Theorem Prover (*http://www.ksl.stanford.edu/software/JTP*) inference engine has been integrated with KAoS. Performance is always an issue in logic reasoning; however, the steady improvement of JTP has led to a dramatic increase in its performance—an order of magnitude or more in some cases—in the last two years. The most time consuming operation in JTP is asserting new information, which happens mostly during system bootstrap. Currently, loading of the KAoS core ontologies takes less than 16 seconds on Pentium III 1.20 GHz with 640 MB RAM. Adding a policy takes usually less than 340ms. Querying JTP about ontology concepts and policies is much faster and takes only a few milliseconds.

Policy representation. In KAoS, policies can express *authorization* (i.e., constraints that permit or forbid some action) or *obligation* (i.e., constraints that require some action to be performed, or else serve to waive such a requirement) for some type of action performed by one or more actors in some situation [1]. Whether or not a policy is currently applicable may be conditional upon some aspect of the situation. Auxiliary information may be associated with a policy, such as a rationale for its existence or a specification of some penalty for policy violation. In contrast to many existing policy systems [*http://www.policy-workshop.org*], KAoS aims at supporting both an extensible vocabulary describing concepts of the controlled environment and also an evolution of its policy syntax. Such features are one beneficial consequence of defining policies within ontologies and using an extensible framework architecture [16].

In KAoS, a policy is represented as an ontology instance[2] of one of the four types of policy classes: positive or negative authorization, and positive or negative obligation. The instance possesses values for various management-related properties (e.g., priority, time stamp, site of enforcement) that determine how the given policy is handled within the system. The most important property value is the name of a controlled action class, which is used to determine the actual meaning of the policy. Authorization policies use it to specify the action being authorized or forbidden. Obligation policies use it to specify the action being obliged or waived. Additionally the controlled action class contains a trigger value that creates the obligation, which is also a name of the appropriate class of actions. Policy penalty properties contain a value that corresponds to a class of actions to be taken following a policy violation.

As seen from this description, the concept of action is central to the definition of KAoS Policy. Typically any action classes required to support a new policy are generated automatically by KAoS when a user defines new policy (usually using KPAT). Through various property restrictions, a given subject of the action can be variously scoped, for example, either to individual agents, to agents of a given class or to agents belonging to a particular group, and so forth. The specific contexts in which the policy constraint applies can be precisely described by restricting values of the action's properties, for instance requiring that a given action be signed using an algorithm from the specified group.

[1] This distinction allows reasoning about actions on policies and the policy framework without resorting to the use of special "metapolicy" mechanisms.

[2] See *http://ontology.ihmc.us/SemanticServices/S-F/Example/* for an example of KAoS policy syntax.

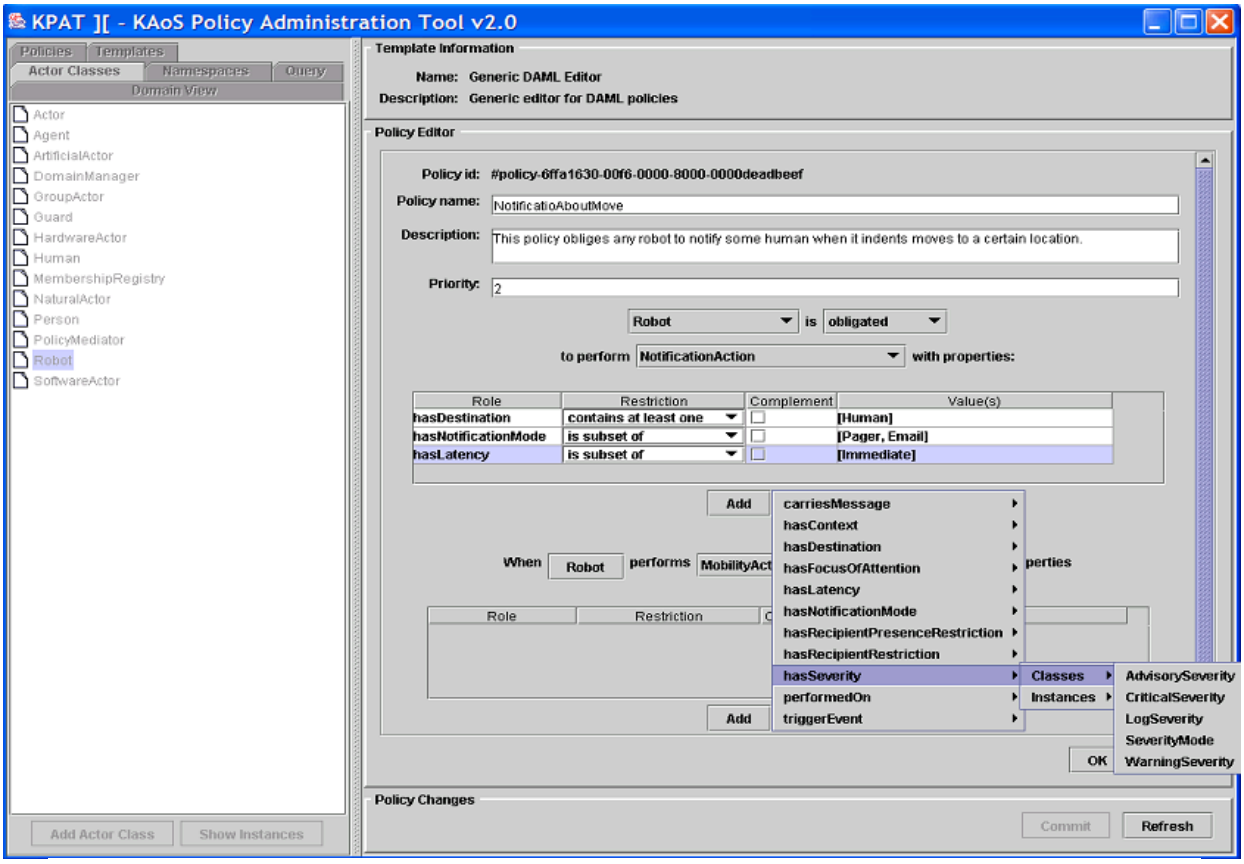

Figure 32 KAoS Policy Administration Tool (KPAT) policy builder interface.

Policy Management

A strength of KAoS is in its extensive support for policy life-cycle management. KAoS hides many elements of complexity of this process from the user. KAoS also provides a sophisticated policy disclosure interface enabling querying about policy impact on planned or executed actions.

Graphical interface to ontology concepts. The KPAT graphical interface to policy management hides the complexity of the OWL representation from users. The reasoning and representation capabilities of OWL are used to full advantage to make the process as simple as possible. Whenever a user has to provide an input is always presented with a complete set of values he can choose from, which are valid in the given context.

As in the case of the generic policy editor shown on figure 2, a user, after selecting an actor for a new policy, is first presented with the list of actions the given type of actors is capable to perform based on the definition in the ontology relating actions to actors by the *performedBy* property. When the user selects a particular action type information about all the properties, which can be associated with the given actions, are presented. For each of the properties, the range of possible values is obtained; instances and classes falling into this range are gathered if the user wants to build a restriction on the given property, thus narrowing the action class used in the build policy to its context.

Policy administration. Each time a new policy is added or an existing one is deleted or modified, the potential impact goes beyond the single policy change. Policy administrators need to be able to understand such interactions and make sure that any unwanted side effects are eliminated. KAoS assists administrators by identifying instances of given types of policy interactions, visualizing them, and, if desired, facilitating any necessary modifications.

One important type of interaction is a policy conflict [4; 18]. For example, one policy might authorize actor A to communicate with any actor in group B while a new policy might forbid actor A from communicating with actor B1, a member of B. In general, if a new policy overlaps in key properties of a subset of controlled actions with an existing policy of a potentially conflicting modality (i.e., positive vs. negative authorization (as in our example); positive vs. negative obligation; positive obligation vs. negative authorization), some means must be used to identify the conflict and to determine, in the area of overlap, which policy takes precedence[1]. If precedence cannot be determined otherwise, KAoS will ask the administrator to determine the appropriate action (figure 3).

[1] If desired, precedence relations can be predefined in the ontology, permitting partially or totally automated conflict resolution.

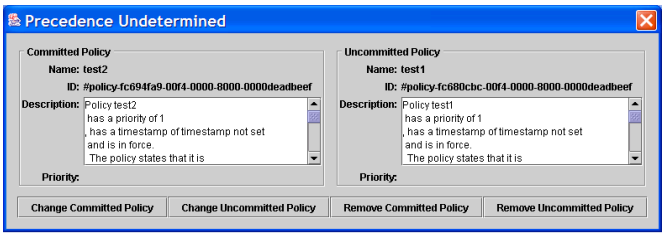

Figure 3. Notification about policy conflict and options available to the administrator.

The following policy actions can be performed on a pair of overlapping policies:

- *Remove Policy*: one of the overlapping policies can be completely removed;
- *Change Priority*: priorities of the policies can be modify so they either do not conflict or they alter the precedence relation[1];
- *Harmonize Policy*: the controlled action of the selected overlapping policy can be modified using an automatic harmonization algorithm to eliminate their overlap; see [4; 18] for details. This required modification of the restrictions in of the policy controlled actions by building either intersection (by using *owl:intersectionOf*) or differences (by using *owl:complementOf*) of the previous ranges in the two conflicting policies.
- *Split Policy*: the controlled action of the selected overlapping policy can be automatically split into two parts: one part that overlaps with the other policy and the other which does not. Then the priorities of these parts can be modified independently. The splitting algorithm is similar to the harmonization and is currently in development.

In the future, a more sophisticated user interface will allow for modification of entire sets of policies at once.

Whereas the goal of policy conflict resolution is to ensure consistency among the policies in force, other forms of analysis are needed to ensure policy enforceability. In some cases, the implementation of policy may be impossible due to prior obligations of the actor or oversubscription of resources. In the future, KAoS will be able to suggest ways of relaxing such non satisfy constraints in certain situations.

In some cases, two complementary policies of the same modality can create unanticipated problems. For example, one policy may prevent communication among actors within domain A while another policy might prevent communication to actors outside of the domain. Though the two policies would not conflict, their combination would result in the inability of actors in domain A to communicate at all. It should be possible in the future to flag these and other situations of potential interest to administrators.

Policy exploration and disclosure. A human user or software component uses KAoS to investigate how policies affect actions in the environment. In general, the answers to these queries are decided by inferring whether some concrete action falls into a category of action controlled by one or more policies, and then determining what conclusions about the described action can be drawn. As part of KAoS policy exploration and disclosure interfaces we provide the following kinds of functionality:

- *Test Permission:* determine whether the described action is permitted.
- *Get Obligations:* determine which actions, if any that would be obligated as a follow on to some potential action or event. For instance, there might be an obligation policy which specified that if an actor were to receive information about a particular topic then the system would be obligated to log or forward this information to some other party.
- *Learn Options:* determine which policy-relevant actions are available or not available in a given context. For example, the actor may specify a partial action description and KAoS would return any missing (required) elements of the action with ranges of possible values—for instance, information about missing credentials.
- *Make Compliant:* transform the action an actor tries to perform from a policy non-compliant to a policy-compliant one by informing it about the required changes that would need to be made to the action based on existing policies. For instance, if the system attempted to send a message about particular subject to a few actors, the list of actors might need to be trimmed to some subset of those actors or else extended to include some required recipients. Or else maybe the content of a message would need to be transformed by stripping off sensitive information, and so forth.
- *Get Consequences:* determines the consequences of some action by observing and investigating possible actions in the situation created by a completion of the considered action(s) to the specified depth (consequences of consequences). This option has many variants currently under investigation.

Adapting policy to legacy systems. When policy leaves the Directory Service, for performance reasons it typically has to map OWL into a format that is compatible with the legacy system with which it is being integrated. KAoS communicates information from OWL to the outside world by mapping ontology properties to the name of the class

[1] We currently rely exclusively on the combination of numeric policy priorities and update times to determine precedence—the larger the integer and the more recent the update the greater the priority. In the future we intend to allow people additional flexibility in designing the nature and scope of precedence conditions. For example, it would be possible to define default precedence over some policy scope based on the relative authorities of the individual who defined or imposed the policies in conflict, which policy was defined first, and so forth.

defining its range as well to a list with cached instances of that class that were in existence when the policy left the Directory Service. A particular system can use the cached instance for its computation; also in any moment it can refresh the list by contacting the Directory Service and providing the name of the range. Alternatively, the Directory Service can push changes to the system as they occur.

Conclusions and Future Work

Whereas most ontologies involved with agent autonomy are concerned with generating plans for what an agent should *do*, KAoS and its ontologies are one of the few that aim to specify how agent behavior should be *constrained*. As to the usefulness of this perspective, Sheridan observes:

> "In democracies specification of 'shoulds' is frowned upon as an abridgement of freedom, and bills of basic rights such as that of the United States clearly state that 'there shall be no law against…', in other words declarations of unconstrained behavior. In such safety-sensitive industries as aviation and nuclear power, regulators are careful to make very specific constraint specifications but then assert that those being regulated are free to comply in any manner they choose.
>
> Vicente and Pejtersen assert that constraint-based analysis accommodates much better to variability in human behavior and environmental circumstance. They make the point that navigating with a map is much more robust to disturbance and confusion over detail than navigating with a sequence of directions" [12, pp. 212-213].

Over the next several months we hope to complete the development of a formal model to describe how a combination of ontology-based inference and decision-theoretic methods can lead to effective autonomy adjustments. We expect many interesting results from the continuation of these studies of the "other side" of autonomy.

References

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

[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. (pp. in press). Berlin, Germany: Springer-Verlag.

[3] Bradshaw, J. M., Sierhuis, M., Acquisti, A., Feltovich, P., Hoffman, R., Jeffers, R., Prescott, D., Suri, N., Uszok, A., & Van Hoof, R. (2003). Adjustable autonomy and human-agent teamwork in practice: An interim report on space applications. In H. Hexmoor, R. Falcone, & C. Castelfranchi (Ed.), Agent Autonomy. (pp. 243-280). Kluwer.

[4] 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,

[5] Bunch, L., Breedy, M. R., & Bradshaw, J. M. (2004). Software agents for process monitoring and notification. Proceedings of AIMS 04.

[6] 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.

[7] Falcone, R., & Castelfranchi, C. (2002). From automaticity to autonomy: The frontier of artificial agents. In H. Hexmoor, C. Castelfranchi, & R. Falcone (Ed.), Agent Autonomy. (pp. 79-103). Dordrecht, The Netherlands: Kluwer.

[8] Feltovich, P., Bradshaw, J. M., Jeffers, R., & Uszok, A. (2003). Social order and adaptability in animal, human, and agent communities. Proceedings of the Fourth International Workshop on Engineering Societies in the Agents World, (pp. 73-85). Imperial College, London,

[9] 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,

[10] 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,

[11] Myers, K., & Morley, D. (2003). Directing agents. In H. Hexmoor, C. Castelfranchi, & R. Falcone (Ed.), Agent Autonomy. (pp. 143-162). Dordrecht, The Netherlands: Kluwer.

[12] Sheridan, T. B. (2000). Function allocation: algorithm, alchemy or apostasy? International Journal of Human-Computer Studies, 52(2), 203-216.

[13] 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,

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

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

[16] 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.

[17] 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,

[18] 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,