Dynamic policy enforcement in JBI information management services with the KAoS Policy and Domain Services

Justin Donnelly[*a], Jacob Madden [a], Alden Roberts [a], Matthew Greenberg [a],
Jeffrey Bradshaw [b], Andrzej Uszok [b]

[a] Lockheed Martin Advanced Technology Laboratories,
760 Paseo Camarillo, Camarillo, CA, USA 93010

[b] Institute for Human and Machine Cognition, 40 South Alcaniz Street, Pensacola, FL, USA 32502

ABSTRACT

English-language policies about the desired behavior of computer systems often suffer from translation errors when implemented with a proliferation of low-level rules governing access control, resource allocation and configuration. To solve this, Dynamic Policy Enforcement systems replace these low-level rules with a relatively small number of semantically grounded, machine-understandable policy statements. These statements use domain terms defined in an ontology that are formally defined so that they can be enforced by the system but also meaningful to human administrators to ensure that they accurately represent organizational policies. In this paper, we describe the application of one such Dynamic Policy Enforcement system, KAoS, to the control of distributed, information-management services defined by the Air Force Research Laboratory's Joint Battlespace Infosphere (JBI) program. Our research allows administrators to define the desired behavior of the participants in the system, both human and software, with one collection of well defined policies. As a result, a single set of tools for the definition, analysis, control, and monitoring of policy can be used to implement access control, service configuration, and service delivery prioritization.

Keywords: dynamic policy enforcement, information management, JBI, KAoS, ontology, OWL

1. INTRODUCTION

Organizational and system policies are generally defined by operational personnel and then implemented by technical personnel interpreting English-language descriptions of the policies. The result is a proliferation of detailed rules for system behavior whose connection to the source policies is difficult to maintain and often poorly documented. Dynamic policy enforcement systems attempt to solve this problem by operating directly on policy statements that are built from semantically grounded terms that correspond in a clear and direct manner with English-language policies. For example, a policy prohibiting personnel with expired security clearances from accessing certain computing resources would traditionally be implemented with a manual or ad-hoc automated process. There would not necessarily be any way to trace why a user was removed from an access-control list based on this policy or to analyze whether it was being implemented effectively and accurately. However, in a dynamic, policy-enforcement system a machine-understandable policy statement would be created and policy enforcement points (PEPs) would deny access to the controlled resources based specifically on the current state of the accessing user's clearance. Logs of this action would clearly indicate why the user was denied access and the same policy statement would allow access to the user in question when the state of their clearance changed back to active.

These flexible, high-level policy statements simplify management while increasing control over subject systems. Ongoing research[+] aims to create policy frameworks and languages that allow policy to be defined and enforced with abstract but formal concepts of the actors and resources in a system.[^] This allows a much more manageable number of rules whose connection to the source policy is clearer *and* whose semantics can be analyzed automatically for conflicts

[*] jdonnell@atl.lmco.com; phone 1-805-484-6100; fax 1-805-484-6195; http://www.atl.external.lmco.com/

[+] For example, the International Semantic Web Conference's Policy Workshop: http://www.cs.umbc.edu/swpw/.

[^] For an analysis of existing frameworks: http://www.l3s.de/~olmedilla/events/2006/ESWC06/ESWC06_Tutorial.html.

and deviations from intent. Some of this research aims to expand the notion of policy beyond security to the most general capability to exert *control* over the behavior of a system. Some of these efforts base their policy representations on Semantic Web languages in order to exploit the powerful set of tools and content available from that research and development community. One such example is the KAoS Policy and Domain Services[1] produced by the Institute for Human and Machine Cognition (IHMC).

Lockheed Martin Advanced Technology Laboratories (LM ATL) and IHMC have been engaged in an extensive program of research and development whose aim is to apply KAoS to the control of a variety of information-management services defined by the Air Force Research Laboratory's Joint Battlespace Infosphere (JBI) program. This program has defined predicate-based publish and subscribe services that provide much more sophisticated access to information than conventional topic-based pub/sub systems, data persistence services, and information transformation services. The goal is to provide a single language and a unified mechanism with which to define and enforce the desired state of and limits on the overall system, despite the significant variety in the functions these services provide and their distributed nature. This unification provides simpler, more intuitive system administration, and is subject to a variety of techniques for analysis and simulation.

This paper describes the results of our work and points out additional avenues of research that could be pursued in this area, some of which is being addressed in ongoing work at LM ATL. Section 2 describes Dynamic Policy Enforcement in more detail, while Section 3 discusses KAoS specifically. Section 4 describes key research results from our efforts to prototype control of JBI services with KAoS. Section 5 describes a future research direction that could exploit explicit policies to reduce risk in systems through analysis and simulation. Finally, Section 6 summarizes our conclusions.

2. DYNAMIC POLICY ENFORCEMENT

2.1 Requirements

Explicit policies can help in dynamically regulating the behavior of complex systems and in maintaining an adequate level of security, predictability, and responsiveness to human control. Policies provide the dynamic bounds within which a system is permitted to function and limit the possibility of unwanted events occurring during operations. By changing policies, system behavior can be adjusted without modifying or re-deploying code. However, traditional policy approaches have been primarily designed for computing environments with a relatively fixed set of resources, users, and services. With networks becoming larger and more dynamic (e.g., Network-Centric Warfare systems or Service-Oriented Architectures), traditional policy approaches become inadequate. The dynamicity, unpredictability and heterogeneity of today's networked environments complicate the design of representations and tools for policy management. Resources are not pre-determined, interacting parties and components are not always known a priori, and mobile users (and potentially mobile code) have varying resource availability depending on the local context. The network topology itself changes as machines go down or come back up, or whenever reconfiguration is necessary due to the rapidly changing nature of the battlefield. Even worse, due to the pervasiveness of computing knowledge among our enemies, the threat of malicious intrusion or attack is ever-present, and could come from anywhere, or be directed at any part of the network. All of these concerns make policy even more valuable than it ever has been in the past and require the ability of policy implementations to have the agility to face a constantly changing modern environment.

The policy management system required to handle these situations needs to use a machine-understandable description of the context where its managed elements execute. It should be able to acquire, reason about, and negotiate the policies that govern behavior in each new context so that appropriate policy decisions can be made and enforced on the fly. To address these issues requires a semantically rich, dynamically adjustable approach to policy representation, specification, analysis, and enforcement. Semantically rich representations permit both structure and properties of the human, situational, environmental and computational elements of complex distributed system and the management operations themselves to be described at a high level of abstraction, thus enabling policy conflict detection and harmonization. Ontology-based approaches rely largely on the expressive features of description-logic languages such as OWL to classify contexts and policies, thus enabling powerful analysis methods and real-time policy conflict resolution to be performed. Description-logic-based capabilities provide efficient, decidable reasoning, while allowing the possibility of additional logical (e.g., the Semantic Web Rule Language (SWRL)) and mathematical extensions (e.g., decision-theoretic methods, Markov models) as required. Thus, changes to policy can be subject to verification and validation.

2.2 Benefits

By representing policies in an explicitly declarative form instead of burying them in implementation code or scattering them in a proliferation of access-control rules, policy-based systems offer opportunities for increased:

- Reusability. Policies encode sets of useful constraints on component behavior, packaging them in a form in which they can be easily reused.
- Reasoning about component behavior. As permitted by policy disclosure policies, sophisticated components can reason about the implications of the policies that govern their behavior and the behavior of other components.
- Extensibility. A layer of basic representations and services can be easily extended to diverse and evolving platforms and sets of operational capabilities.
- Verifiability. Theorem provers and formal methods can ensure that program behavior which follows the policy will also satisfy many of the important properties of reactive systems such as liveness, recurrence, and safety invariants. Additionally, it is tremendously valuable that administrators can easily verify that semantically grounded system policies accurately implement organizational policy.

2.3 Contrasted with Traditional Access Control Systems

Traditional access control systems tend to provide useful abstraction hierarchies of the actors in a system (roles), but no complex groupings of roles or corresponding abstractions of the possible actions or context of the actions. Consider, for example, most file systems. User groups can be established, but the action concepts are too few (read, write, execute), the targets are too many (a rule for every file) and there's no provision for context. Ontologically based policy management systems provide meaningful abstraction hierarchies for actor, action, and context (e.g., system state). A small number of policies built around abstract concepts can replace 1000's of low-level rules about system behavior. Note that these abstractions allow multiple policies to apply to a given action. This requires reasoning about the applicability of policies, and can potentially lead to confusion about the relative priority of policies. But, consider the situation where we want to allow officers of coalition militaries to access military reports, but deny them access to satellite imagery reports, unless the report covers their area of responsibility. In a policy-based system, three policies represent this, using the terms above. In most traditional systems, this would be implemented by a proliferation of rules, hopefully with some documentation kept somewhere as to why the group IraqiMilitaryOfficers were allowed to access the resource /Reports/Satellite/Imagery/CentralIraq/Najaf but not any other resources in /Reports/Satellite/Imagery/.

3. KAOS POLICY AND DOMAIN SERVICES

KAoS provides policy, domain and other services for a wide variety of agent, robotic, and distributed-computing platforms. Its policies are defined as instances of Web Ontology Language (OWL) classes defined in a set of extensible policy ontologies. Extensions to OWL's description-logic-based mechanisms can be introduced when needed (e.g., role-value maps). KAoS provides sophisticated query and analysis mechanisms, including policy deconfliction, and features a policy dissemination and decision-making infrastructure that is distributed, efficient and pluggable. While initially oriented to the dynamic and complex requirements of software agent applications, KAoS services have also been adapted to general-purpose grid computing, CORBA, and Web Services environments. Figure 1 presents basic elements of the KAoS framework. Framework functionality can be divided into two categories: generic and application/platform-specific. The generic functionality includes reusable capabilities for:

- Creating and managing the set of core Ontologies
- Storing, deconflicting and querying
- Distributing and enforcing policies
- Disclosing policies.

For specific applications and platforms, the KAoS framework can be extended and specialized by defining new ontologies describing application- and platform-specific entities and relevant action types.

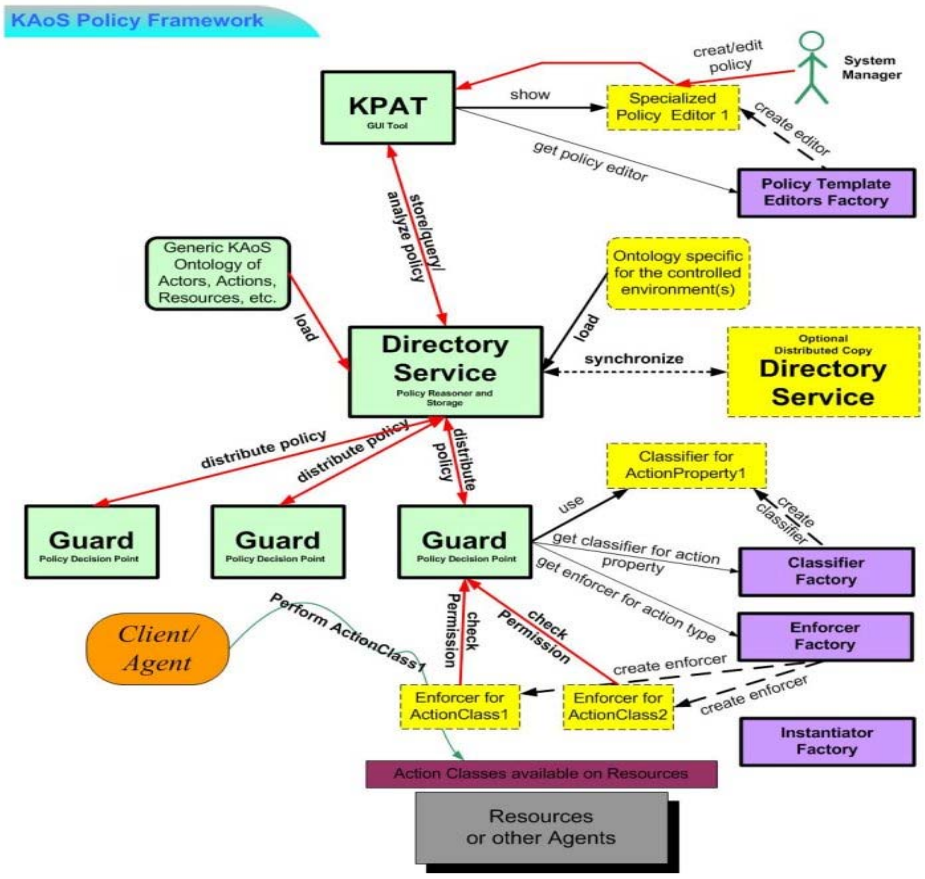

Figure 1. The KAoS Policy and Domain Services framework.

3.1 KAoS Policy Representation

The KAoS services use a core set of ontologies expressed in OWL as a basic vocabulary for actors, actions, and context to be used during policy creation. These concepts can be extended with domain-specific ontologies to further describe policies. Figure 2 displays an example policy using both the core ontologies (Policy.owl, Action.owl) and a domain-specific ontology (Domains.owl) and shows the description-logic approach used by KAoS for building policies. The OWL semantics used by KAoS enable using an inference engine to reason about policy, perform conflict detection, and conduct policy conflict harmonization. In addition, subsumption and instance-classification algorithms developed within KAoS are able to take advantage of the policy representation.

A KAoS policy's OWL definition is an instance of one of four basic policy classes corresponding to the modality of the policy:

- PositiveAuthorization
- NegativeAuthorization (the instance in Figure 2)
- PositiveObligation
- NegativeObligation

The property values of a policy instance specify management information (e.g., priority, site of enforcement, and timestamp). A policy instance's "controls" property references a custom action class that is created using OWL restrictions to narrow the scope of the action's properties. This action class can use any available OWL construct (e.g., range of actors performing the action) and policy consequently can contain arbitrarily complex definitions of a situation. Thus, KAoS represents the policies without conditional rules, relying instead on the context restrictions associated with the action class to determine policy applicability in a given situation.

```
<?xml version="1.0" ?>
<!DOCTYPE P1 [
  <!ENTITY policy "http://ontology.ihmc.us/Policy.owl#" >
  <!ENTITY action "http://ontology.ihmc.us/Action.owl#" >
  <!ENTITY domains "http://ontology.ihmc.us/ExamplePolicy/Domains.owl#" >
]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.owl.org/2001/03/owl+oil#"
  xmlns:policy="http://ontology.ihmc.us/Policy.owl#">

<owl:Class rdf:ID="OutsideArabelloCommunicationAction">
  <owl:intersectionOf rdf:parseType="owl:collection">
    <owl:Class rdf:about="&action;NonEncryptedCommunicationAction" />
      <owl:Restriction>
        <owl:onProperty rdf:resource="&action;#performedBy" />
        <owl:toClass rdf:resource="&domains;MembersOfDomainArabello-HQ" />
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&action;#hasDestination" />
        <owl:toClass rdf:resource="&domains;notMembersOfDomainArabello-HQ" />
      </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<policy:NegAuthorizationPolicy rdf:ID="ArabelloCommunicationPolicy1">
  <policy:controls rdf:resource="# OutsiteArabelloCommunicationAction " />
  <policy:hasSiteOfEnforcement rdf:resource="&policy;ActorSite" />
  <policy:hasPriority>10</policy:hasPriority>
  <policy:hasUpdateTimeStamp>446744445544</policy:hasUpdateTimeStamp>
</policy:NegAuthorizationPolicy>
</rdf:RDF>
```

Figure 2. Example KAoS policy.

3.2 KAoS Policy Conflict Detection and Prioritization

Changes and additions to the policies in force in a system will typically come from multiple administrators. Consequently, conflicts are inevitable so KAoS uses logical inference to detect and handle three different types of modality conflict: positive versus negative authorization (i.e., being simultaneously permitted and forbidden from performing some action), positive versus negative obligation (i.e., being both required and not required to perform some action), and positive obligation versus negative authorization (i.e., being required to perform a forbidden action). The policy-conflict detection and harmonization algorithms within KAoS allow policy conflicts to be found and resolved even when the actors, actions, or targets of the policies are specified at very different levels of abstraction. KAoS detects potential conflicts between policies at specification time, whenever a user tries to add a new policy to the Directory Service. The KAoS algorithm for policy conflict detection is able to detect conflicts between policies by relying on algorithms implemented as extensions to Stanford's Java Theorem Prover (JTP)[2], which is integrated into KAoS. The engine identifies policy conflicts by using the subsumption mechanisms among relevant classes.

KAoS relies solely on numeric priority values and timestamps to determine policy precedence, with newer and higher priority value policies having higher precedence. However, the vision for KAoS is to incorporate various factors for the precedence determinations, such as relative role levels of actor classes, policy scope, and domain superiority. With more policies in a system, it will become necessary to have more robust precedence-determination mechanisms in order to allow users to maintain coherent control over system behavior. KAoS also provides a mechanism for performing policy disclosure, whereby you provide an action instance to check policy against and KAoS returns whether or not an action is authorized. This provides an interactive way to evaluate the effects of policy.

4. CONTROL OF JBI INFORMATION MANAGEMENT SERVICES

4.1 JBI Information Management Services

The Air Force Research Laboratory Information Directorate's (AFRL/IF) JBI research and development program is based on two summer-study reports published by the Air Force Science Advisory Board. These two reports, Information Management to Support the Warrior (1998) and Building the Joint Battlespace Infosphere (1999) describe a combat-information-management system that, in accordance with policy, allows users to provide, discover and exchange the information required to perform their operational duties during crisis or conflict. The goal of the JBI program is to develop technology that fulfils this vision. JBI technology enables and manages the distribution of information produced by a wide variety of sources, transforms it as necessary and delivers it in an appropriate form to those who need it.

4.2 Dynamic Policy Enforcement in JBI Services

The goal of the research described in this paper was to bring the benefits of dynamic policy enforcement to the configuration and control of JBI information-management services. These services include, among other things, a publish and subscribe broker, an information repository, and an information transformation layer. The functions provided by these services are designed to be complementary but are very different in how they operate. The publish/subscribe broker is designed to be continuously responsive to new publications, routing them to subscribers whose predicates match the content of the publications, while the information repository satisfies demands only in response to specific requests, and the transformation layer consists of semi-independent information-transformation components that have their own publish and subscribe streams. These services need to be subject to very different kinds of control. For example, providing simple configuration values is sufficient for most kinds of adjustments to the behavior of the broker, while more subtle rules need to be enforced to control the behavior of a multitude of independently executing information-transformation components.

At the heart of our effort was the development of a system bringing together implementations of standardized JBI services with dynamic policy enforcement based on KAoS. The principle technique was to modify existing service implementations by replacing their existing, custom implementations of access control, service prioritization, and configuration with generic interfaces to Policy Decision Points (PDPs). These interfaces could then be instantiated with simple mechanisms replicating the previous behavior or with KAoS-based components making decisions based on the policies in force. Figure 3 illustrates this coupling of existing JBI services with KAoS policy services. In the diagram, "J-DASP" refers to one of the projects under which this research was performed. The key pattern to note in the diagram is the coupling of the two sets of services through the PDP interfaces (labeled, for example, "Access Control API") and "enforcers" that implement the interfaces. Enforcers[*] are a KAoS concept and function as the boundary point between the KAoS services and the system under control. The enforcers plug into KAoS and can query KAoS for information about policy decisions or can be notified of actor obligations via an event-based mechanism. Also of interest in this discussion is the Platform State Classifier, described in the next section.

4.3 System State Classification

The benefits of dynamic policy enforcement discussed in this paper include:

- The use of abstract policy representations to replace a proliferation of low-level rules with a smaller number of semantically grounded policies;
- The ability to have policy decisions subject to context.

One of the key kinds of context used in our work is the current state of the services under control. We use instrumentation of the various services as well as bandwidth and processor utilization statistics to influence the application of policies. Our goal was to define and enforce policies that would state, for example, that service logging should be performed at the INFO level, unless the broker service is heavily burdened, in which case we want to log at the WARN level. Such a policy could be used as an element of a strategy to gracefully degrade services when under heavy load.

---

[*] Enforcers are commonly referred to as "Policy Enforcement Points" (PEPs) in the policy literature.

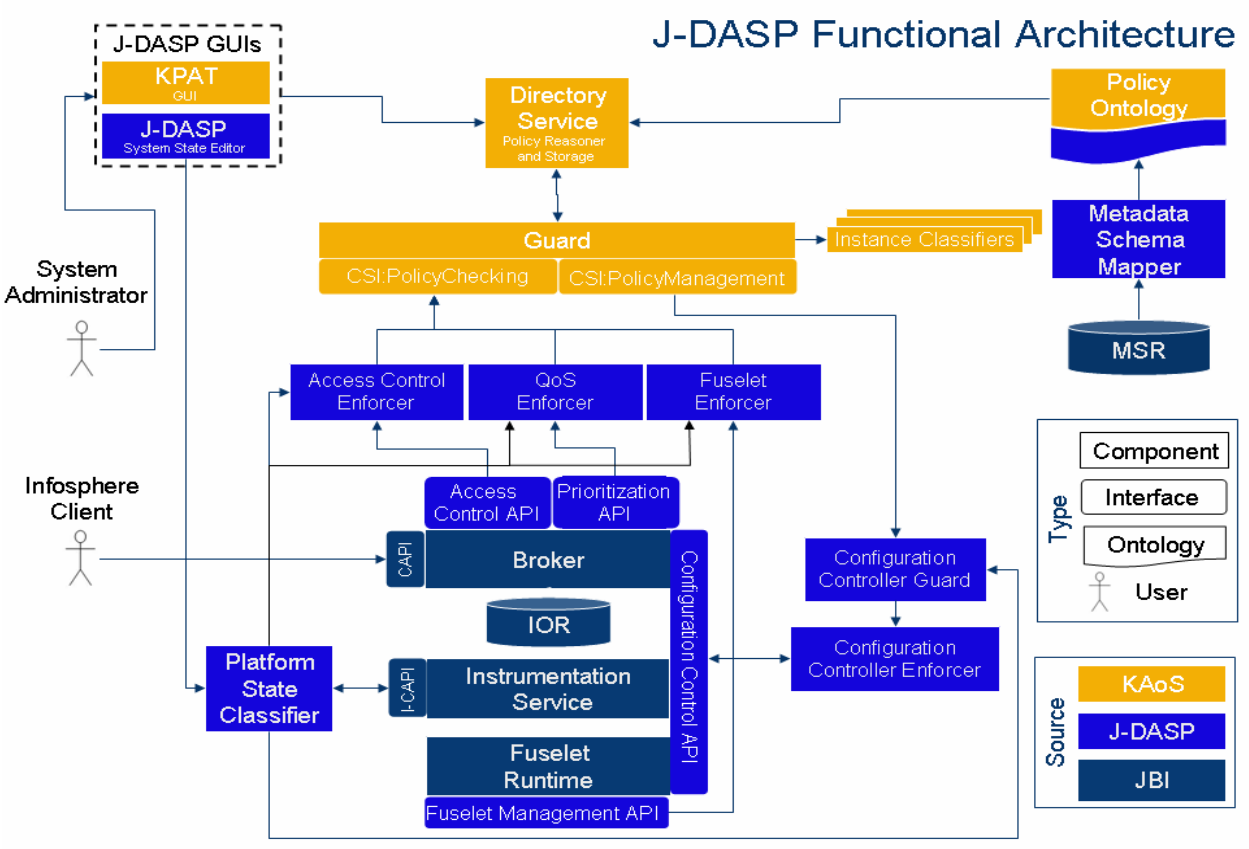

Figure 3. Architecture diagram for combined JBI and KAoS services.

However, to preserve the first benefit above, these statistics about resource utilization and service employment must be made available in a relatively abstract form. Our approach was to define finite sets of system and service states. These states allow policy definitions to be sensitive to current conditions and to still be abstract and semantically meaningful. So instead of defining our policy as "log at WARN level if the number of broker service deliveries is less than 50 / second" we define it as "log at WARN level if the system is in the BrokerHeavilyBurdened state". The former is arbitrary and unclear. Other policy definitions that want to be sensitive to the performance of the broker might choose 40 / second or 75 / second, and when we read the policy, it's not clear if the author considered 50 service deliveries per second to be a mild condition or a very severe condition. The latter policy uses a semantically meaningful term that can be defined once, used in many policies, and adjusted over time to reflect changes to the system and our understanding of what are good and bad states.

Figure 4 shows the tool we developed to define these abstract system states. It features a conventional user-interface mechanism for defining states based on one or more numerical comparisons. One very innovative feature, however, is the live display of instrumentation streams from the current system or a simulation. This allows an administrator to get some insight into what the nominal behavior of the system is, rather than having to guess what constitutes "normal" performance. These system state definitions are then used by a component called the Platform State Classifier, which consumes instrumentation streams and live system statistics, makes decisions about the current states of the system, and then shares these states with policy decision points via a distributed, event-based mechanism.

4.4 Monitoring the Effects of Policy

The application of KAoS to existing JBI services allows us to control access to resources (who can publish, subscribe, query, or deploy information transformation components as well as what they can perform these actions on and when and under what conditions), the configuration of logging and data retention, the quality of service delivered to each client (described in the next section. To help us understand the effects of dynamic policy enforcement over all these

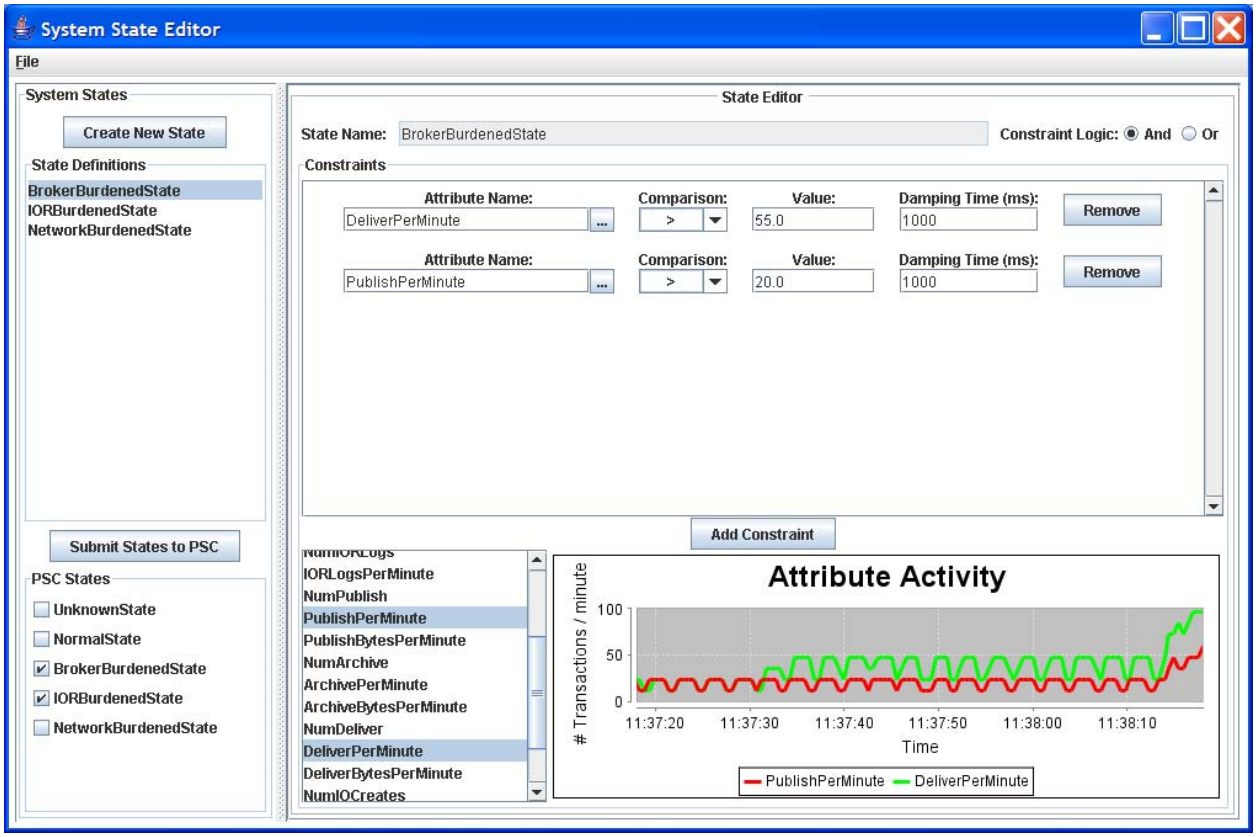

Figure 4. Screenshot of the LM ATL system state definition prototype.

behaviors, Lockheed Martin ATL implemented a configurable system monitor that exploits available instrumentation streams in order to visualize in real time the state of the system. Figure 5 is a screenshot of this tool. Key aspects of the demonstration shown in this screenshot are the response of the logging system in the lower graph at 12:24:00 in response to a high system load, seen at the same point in the upper graph, and the independence of the number of publications and the number of deliveries of those publications ("SubscriberDelivery"), which in the demonstration are controlled by changes to the access-control policies for the sensitive information being published.

4.5 "Soft" Quality of Service

One of the more subtle and complex applications of policy in our work has been the development of a mechanism to provide a policy-driven Quality-of-Service (QoS) capability to the publish/subscribe and repository query services. Quality-of-Service refers to the capability of a networked system to provide different classes of service to selected network traffic. We refer to this as "Hard" QoS, which provides guarantees on bandwidth and latency of certain traffic. However, a highly dynamic and heterogeneous system, such as the JBI services in our work, may not be able to provide such guarantees. Instead, we have focused on providing "Soft" QoS, which we define as providing best-effort prioritization of higher classes of traffic, while balancing these efforts with efficient utilization of resources (e.g., maximizing total throughput and avoiding starvation of low-priority traffic). QoS attributes include latency between publication and reception (or query request and result delivery) of an information object as well as the frequency of information object delivery. QoS-driven resource decisions determine how latency and frequency degrades for different sources, destinations, information object types, payload characteristics, and platform conditions.

Our implementation of this concept starts with another application of the pattern of defining a manageable set of abstract concepts. In this case we defined an ontology with four classes of service: Very High, High, Medium, and Low priority. The dynamic nature of ontologies and our policy administration system allow these classes to be modified at any time to include more or fewer service classes. We then can define policies that call for a particular class of service for particular

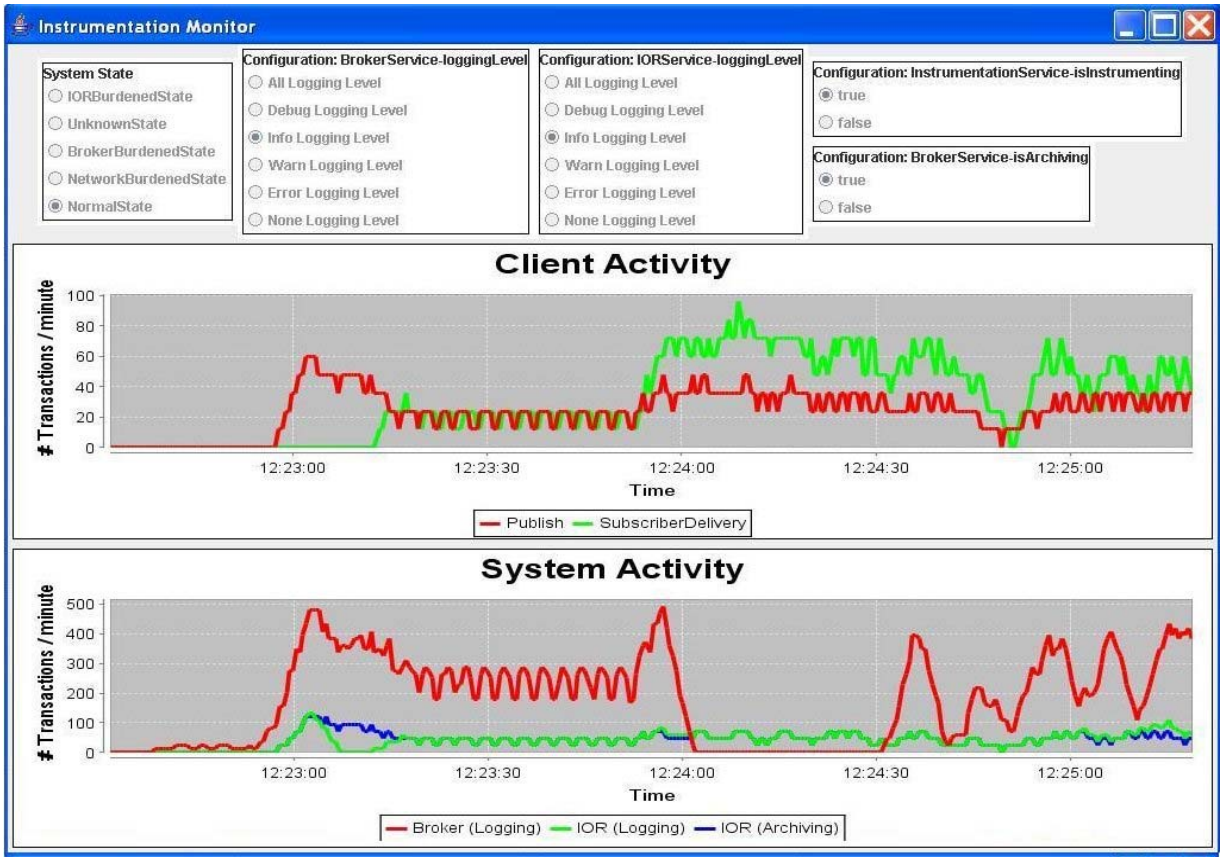

Figure 5. Screenshot of LM ATL instrumentation monitor.

operations. For example, a policy may declare that publications by the Secretary of Defense be given High priority, while another, higher priority policy may state that monthly report publications should be treated with Low priority. At the time of a given service request, such as the publication of a piece of information, we then examine all of the policies that apply to that action. There are two approaches we offer in considering these policies. Either the highest priority policy can govern (here we refer to the priority of the policy itself, *not* the class of service specified by the policy) or all applicable policies can be input into a function that produces an average of numerical values for our four classes of service, weighted by the priority of the policies. So if both of the policies above were in force when the Secretary of Defense published a monthly report, this would either be given Low priority if only the higher priority policy is applied, or a class of service between High and Low, based on the function that combines the two relevant policies.

5. SYSTEM RISK MODELING AND ANALYSIS

In addition to LM ATL's efforts to prototype policy-based control of JBI services, we have performed additional research into the potential to exploit our policy representations to perform novel methods of risk analysis. Most processes for evaluating, identifying, and analyzing risk in systems are based on a formal process that often involves system-specific questionnaires and other paperwork (e.g., the NIST methodology or the INFOSEC Assessment Methodology (IAM)). While these provide critical insight into the innate risk of systems, their formality makes them less agile and reactive to changes in the system. Policy-based systems are meant to be more reactive and flexible. By drawing on the best parts of various methodologies of system modeling and analysis, we hope to create a method of assessment that provides accurate risk assessment of policy-based systems while minimizing user intervention.

5.1 Modeling Risk in Known Systems

The modeling process takes the identification of three key elements of the system: Actors, Resources, and Channels:

- Actors: the clients of the system (trusted, untrusted, and even anonymous).
- Resources: units of information (e.g., documents), processing power, or services.
- Channels: the means by which an Actor or Resource accesses another Actor or Resource (these can range from a direct terminal or hard-line to an unencrypted connection across the Internet).

For the purposes of initial modeling, these three elements are sufficient. The task of creating a model that accurately expresses the behavior of a system requires a great deal of knowledge about the system. While systems administrators might attempt to create this model by hand, this task would often be overwhelming due to the number and distribution of resources in some systems. An alternative is to model the system using the policy that governs it. Because a dynamic policy enforcement framework enumerates the necessary relationships explicitly, generating the model automatically is possible. This model can then be inspected and updated by a human administrator as necessary.

The next step in modeling the system involves identifying the risk characteristics associated with each component. The security characteristics of each element need to be defined:

- Actors: must be annotated with level of trust, accounting for the trust in the particular client and the risk that the account is being spoofed or has been otherwise compromised (e.g., an Actor in the field versus an Actor in known safe location).
- Resources: must be annotated with level of sensitivity (e.g. critical resource or nominal and lesser important resource).
- Channels: must be annotated with their level of security (probability of being compromised).

The manual ranking of every element in a system, even for small systems, requires a large effort. A better approach would be to automatically assign generic values to a majority of the elements based on type. The initial values can then be tweaked by hand based on the operational factors at work within the system. The automatically generated values should tend toward a cautious weighting, though, allowing the administrator to tweak values to remove false positives.

5.2 Model Analysis

After the model is established, there are a variety of methods for characterizing the risk of the system as a whole. While it is generally true that the security of a system is only as strong as its weakest link, for distributed systems governed by comprehensive policy this generalization is less apt, as these systems are more resilient to a single element being compromised. Nonetheless, the intention of this analysis is to identify unnecessary risk in the system to help policy designers become aware of areas that might require policy additions and modifications.

The most rudimentary analysis that can be done is simply identifying sets of three in which one of the elements varies greatly from its connected elements. There are two major issues that this first pass will identify. The first is changes to the policy that will loosen restrictions without increasing risk, for example, if a policy only allows encrypted connections from untrusted users on publicly available documents, then removing the encryption requirement does not increase the risk but does eliminate a superfluous policy. The other is situations in which a critical document is supplied via an insecure channel or to an untrusted user. In this case, the innate risk of the system is greatly increased and the policy needs to be modified to eliminate this risk if it is not absolutely necessary. Beyond this first layer of interaction, there is an additional need to identify secondary (and nth order) channels within the system. For instance, a policy that allows a user to connect unencrypted to a system and then create a new encrypted connection from that system to a resource that requires a secure connection circumvents the intention of security policy. Thus, analysis can be done on an arbitrary path through the graph. These two examples only scratch the surface of the potential analysis that can be accomplished given a comprehensive model of the system and the characteristics of its elements.

5.3 Policy Verifiability

As opposed to behavior that is implicit in application code, explicitly declared policies support important types of policy analysis. First, we can validate that the policies are sufficient for an application's tasks by leveraging automated theorem-provers and human expertise. Second, there are methods to ensure that program behavior that follows the policy will also satisfy many of the important properties of reactive systems: liveness, recurrence, and safety invariants. Liveness is the property of a system where under a certain set of conditions intended behavior will eventually occur. By

proving that a system upholds the liveness property, you are determining that the policy does not preclude the system from accomplishing its intended purpose. This is most fundamentally realized as the ability to guarantee that within a given state its internal computation will result in a transition. Recurrence is the property of a system that given a state with the same inputs the system will react in the same way. Essentially this is a proof of predictability: will the system react consistently to consistent stimuli? While recurrence as a concept is very important, in practice it is often taken for granted and not rigorously proven. An essential task in the design of any safety-critical system is to identify the safe states. A system is only safe if it is always in a safe state. Safe states can be determined by state functions. A state function that is intended to return true in every state that the system can reach is called an *invariant* (its truth never varies). Safety requirements can be represented by invariants. A system that is always in a safe state is said to satisfy its safety invariant. A system that can reach a state where the safety invariant is false is said to violate the invariant (it can enter an unsafe state). Safety requirements for a system are represented by safety invariants that characterize safe system states that never transition to unsafe states. Therefore, this involves identifying the safe and unsafe states of the system. Using logic-based policy can ensure that certain state transitions never occur and thus uphold certain safety invariants. For example, the policies in force could help guarantee that the system never enters a deadlock situation.

These properties are essential to the creation of a policy-based system that functions and more importantly accomplishes its goals. After all, a system that never deadlocks but also never achieves anything is not useful. The outright logical proof of these attributes for an arbitrary system is not a simple task. An increasingly common method is called Spinning[3], which requires a model of the system and the definition of claim in the PROMELA language[4]. The SPIN tool exhaustively enumerates every path across the model checking for violations of the claims enumerated. The reliance on SPIN presents two problems. First, the system must be modeled in a way that is amenable to the proof process. There is a great deal of research into how to optimize the models for a specific purpose, but generally, it is not uncommon for a SPIN model to become computationally intractable if improperly specified. Second, while SPIN automatically checks the liveness and safety of a given system, with built-in proofs, recurrence and other properties require more advanced PROMELA concepts.

An alternative to the approach discussed above is to use the available information to create an approximation of the proofs generated by Spinning. Essentially, if over the course of a large number of transitions (e.g., 100,000) from state A with an input of B resulted in state C, an empirically grounded claim that the system is recurrent for that case is an acceptable approximation. While not a proof, simulating various cases and repeatedly observing recurrent behavior is sometimes the only option and can help increase confidence that the deployed system will behave as expected.

5.4 Policy Decision Log Analysis

In addition to analysis of the policies themselves, there is the opportunity to perform analysis after policy decisions have been made using logs of policy decisions. KAoS generates decision logs based on each of its decisions tracking the time, decision, actor, action, and policy ID. By manually inspecting whether the negative policy decisions correspond to a certain actor or action, administrators can determine if there is a malfunction in the current policy set. Although, this mechanism is useful, it is very labor intensive. A possible extension includes providing an automated analysis engine that constantly searches the log files for interesting patterns. A library of patterns could be provided that allows the user to select which are relevant in the operational domain. In addition, for the most flexibility, the pattern library should be open to additions, possibly authored by users for their specific purposes. Some example patterns include an actor class always being denied a certain action in all situations or all actor classes being denied a certain action or set of actions.

Administrators could then examine the decision-log analysis and decide whether the behavior is consistent with the policy set and the real-world security requirements. Such analysis would also be of use after simulations of predicted system behavior in a test environment. By defining certain desired invariants, such as the expected outcome of key policy decisions, the desired balance between false positives versus false negatives, or the risk associated with certain types of decisions, very informative analysis can be presented to administrators. Figure 6 is a screenshot of a prototype created by LM ATL to illustrate this sort of analysis.

6. CONCLUSIONS

The application of dynamic-policy-enforcement techniques has proven very successful in our prototypes. With a relatively modest software-development effort, much of which was devoted to architecting the services under control to

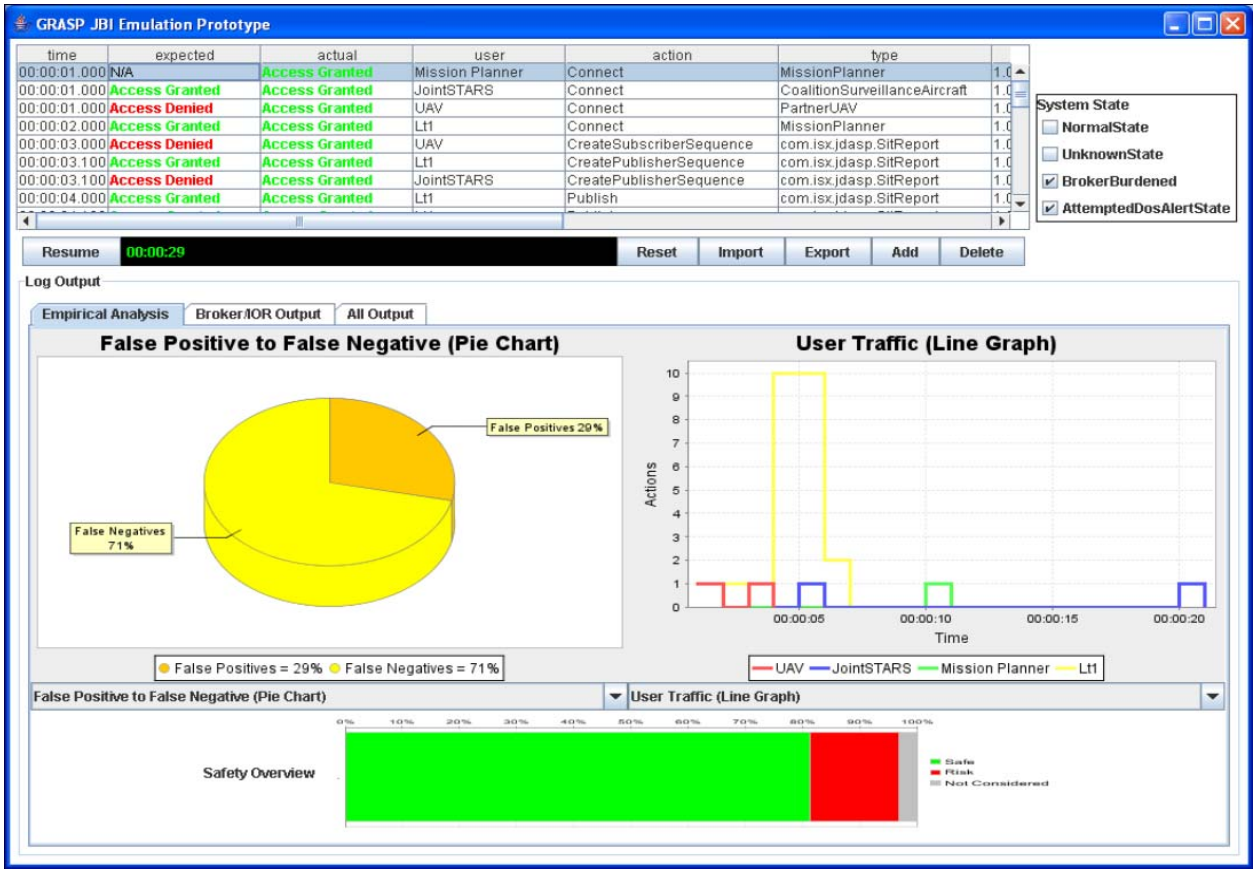

Figure 6. Screenshot of log analysis prototype.

allow for pluggable implementations for access control, configuration control, and service prioritization, we have provided a unified language and set of tools for the specification of a wide variety of high-level rules that govern the behavior of a variety of services. Additional research still needs to be done to improve the ease with which policies can be created and to visualize the relationships between the policies in force in order to make it easier to understand the current state of the policy system. Also, new avenues of research are available to exploit this kind of system by analyzing the policies in force or their effect on live as well as simulated systems.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the support of the Air Force Research Laboratory, especially James Hanna, Chriss Vincelette, and Vaughn Combs who provided oversight and design contributions. The work described herein was performed under the following contracts: J-DASP: JBI Dynamic Administration and System Policy (FA8750-05-C-0198) and GRASP: Generic, Risk-Adaptive Security Policy (FA8750-06-C-0152).

REFERENCES

1. A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. R. Breedy, L. Bunch, M. Johnson, S. Kulkarni, J. Lott, “KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement”, *Proceedings of Policy 2003*, 2003.
2. R. Fikes, J. Jenkins, and G. Frank, "JTP: A System Architecture and Component Library for Hybrid Reasoning." *Proceedings of the Seventh World Multiconference on Systemics, Cybernetics, and Informatics*, 2003.
3. G. J. Holzmann, The Model Checker SPIN”, *IEEE Transactions on Software Engineering.* **23, 5**, 279-295, 1997
4. Spin Online References, http://www.spinroot.com/spin/Man/.