DAML Reality Check:
A Case Study of KAoS Domain and Policy Services

A. Uszok, J. M. Bradshaw, P. Hayes, R. Jeffers, M. Johnson, S. Kulkarni, M. Breedy, J. Lott, L. Bunch

Institute for Human and Machine Cognition (IHMC), University of West Florida,
40 S. Alcaniz, Pensacola, FL 32501
{auszok, jbradshaw, phayes, rjeffers, mjohnson, skulkarni, mbreedy, jlott, lbunch}@ai.uwf.edu

Description-logic-based knowledge representations and reasoning methods are being used increasingly as the basis for semantically-rich software services. Using such representations and reasoning methods in comprehensive applications is among one of the best ways to identify and understand gaps and limitations. KAoS domain and policy services, which rely extensively on a DAML-based ontology, are used as a case study to investigate an in-depth application of DAML. In this context, we explore the current limitations of DAML semantics. We also describe our observations about the place of ontology description at the heart of KAoS services, and outline requirements for effective interfaces for humans and for translation to and from efficient programming environments. Next, we present the problems implicated by development of ontology descriptions in a distributed dynamic environment. Finally, we assess the utility and maturity of available tools such as editors, programming libraries and inference engines.

1 Introduction

DAML (*www.daml.org*), OWL (*http://www.w3.org/2001/sw/WebOnt*), and other increasingly popular description-logic-based representations [1] seem to be a natural choice to support the development of the current generation of semantically-rich software services and intelligent systems [4].[1] The KAoS Policy and Domain Services framework [2, 3, 5, 14] is an interesting example of this trend. By investigating its design, development, and application, we can learn much about the current state of description-logic-based representations, tools, and technology—their strengths, their gaps, and their limitations.

The implementation of the KAoS framework proved to be a challenging task and required integration of the scarce existing DAML and description logic tools—an overview of KAoS is provided in section 2. Development of ontologies necessary for KAoS also provided valuable insight. Sections 3 and 4 describe how ontology development is handled in KAoS and various issues in policy representation relating

[1] OWL Full actually goes beyond description logics in its expressiveness, however as of this writing there is no reasoning engine that supports every feature of OWL Full.

to DAML. The description logic reasoning capabilities integrated into KAoS facilitate analysis of the relations among controlled policy elements and policies themselves, allow for the design of a flexible policy allocation and distribution methodology, and provide the foundation for the implementation of a sophisticated policy query system. Issues relating to the integration of reasoning capabilities are discussed in section 5. Section 6 describes issues relating to the central role of ontologies in the KAoS architecture, both as the internal glue linking its framework components and also as the way in which the KAoS system interacts with other systems. We conclude with a few parting observations (section 7).

2 Case Study Overview: KAoS Policy and Domain Services

KAoS is a collection of componentized services compatible with several popular agent platforms, including Nomads [13], the DARPA CoABS Grid [9], the DARPA ALP/Ultra*Log Cougaar agent framework (*http://www.cougaar.net*) and CORBA (*http://www.omg.org*). The adaptability of KAoS is due in large part to its pluggable infrastructure based on Sun's Java Agent Services (JAS) (*http://www.java-agent.org*). 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.

Under DARPA and NASA sponsorship, we have been developing the KAoS Policy and Domain services to increase the assurance with which agents can be deployed in a wide variety of operational settings. In conjunction with Nomads strong mobility and safe execution features, KAoS services and tools allow for the specification, management, conflict resolution, and enforcement of policies within the specific contexts established by complex organizational structures. *KAoS domain services* provide the capability for groups of software components, people, resources, and other entities to be structured into organizations of 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. Policies are represented in DAML+OIL as ontologies. The KAoS Policy Ontologies (KPO) distinguishes between *authorizations* (i.e., constraints that permit or forbid some action) and *obligations* (i.e., constraints that require some action to be performed, or else serve to waive such a requirement).

Figure 1 presents basic elements of the policy framework. Framework[2] 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 enforcement policies;

[2] Figure 1 emphasizes infrastructure supporting the specification and use of authorization policies. There are additional components to support obligation policies and other aspects of the system but they were omitted from the picture for simplicity; they do not introduce any new aspects relevant to the issues discussed in this paper.

- Disclosing policies.

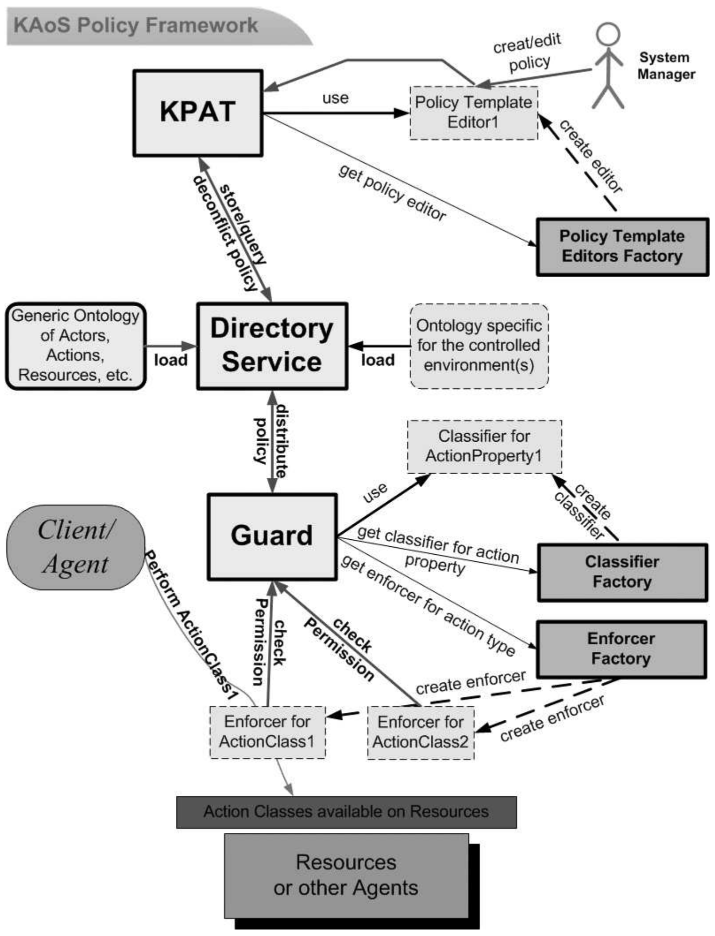

KAoS Policy Framework
creat/edit policy
System Manager
KPAT
use
Policy Template Editor1
create editor
get policy editor
Policy Template Editors Factory
store/query deconflict policy
Generic Ontology of Actors, Actions, Resources, etc.
load
Directory Service
load
Ontology specific for the controlled environment(s)
distribute policy
Classifier for ActionProperty1
use
create classifier
Client/ Agent
Guard
get classifier for action property
Classifier Factory
get enforcer for action type
Enforcer Factory
Perform ActionClass1
check Permission
check Permission
create enforcer
create enforcer
Enforcer for ActionClass1
Enforcer for ActionClass2
Action Classes available on Resources
Resources or other Agents

Fig. 1. Selected elements of the KAoS Policy and Domain Services framework.

For specific applications and platforms, the KAoS framework can be extended and specialized by:

- Defining new ontologies describing application-specific and platform-specific entities and relevant action types;
- Creating plug-ins for:
  - Policy Template and Custom Action Property editors;
  - Enforcers controlling, monitoring, or facilitating general or specific actions;
  - Classifiers to determine if a given instance is in the scope of the given class.

Each of these components will be explained in more detail in the sections below.

3 Ontology Creation and Management

As already indicated, DAML ontologies are central to the KAoS system. Although the DAML specification was relatively stable when KAoS services began to incorporate it, it was not yet well tested. The development process revealed some limitations of DAML, which are presented in the following sections of the paper. We are currently preparing to transition KAoS to OWL—the W3C-approved evolution of DAML—in the near future as soon as tools are sufficiently mature. However, we do not expect it to eliminate these limitations.

The extensibility of KAoS for different applications and platforms requires ontological development to be modular and incremental. The generic ontologies are also divided into different namespaces to facilitate management. Unfortunately, this ruled out the use of DAML editors then available, including those listed on *http://www.daml.org/tools,* such as OilEd or OntoEdit, as well as Protegé (*http://www.ai.sri.com/daml/DAML+OIL-plugin*). None of these editors was capable of linking multiple ontology files together through the use of the *daml:import* property.

3.1 Core KAoS Policy Ontologies

The current version of the core KAoS Policy Ontologies (KPO) defines basic ontologies for actions, actors, groups, places, various entities related to actions (e.g., computing resources), and policies. It includes about 80 classes and 40 properties.

The 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 a joint goal that all actors possess, or a given place where various entities may be currently located).

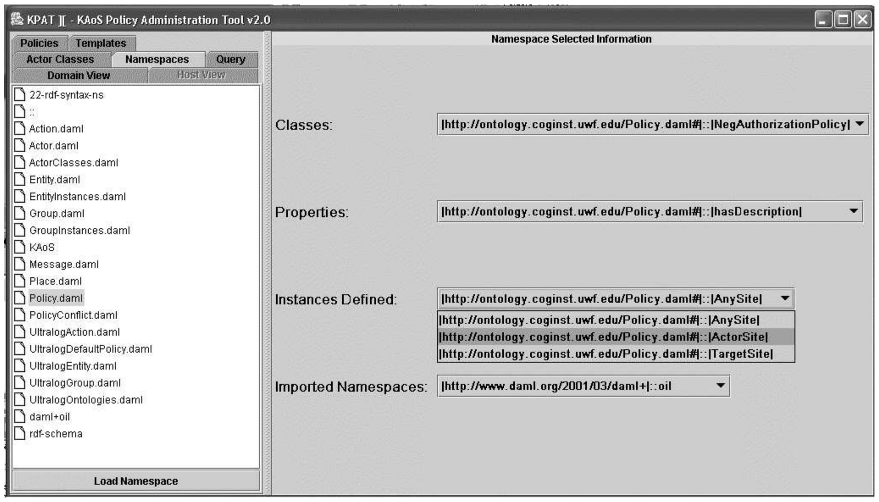
We have found that developers sometimes do not clearly distinguish between those policies that apply to each member of a group individually and those that apply to the group as a whole. For example, a policy that restricts the use of network bandwidth to 50% will be enforced differently if it is intended that each actor in a group will be allowed up to 50% of the network than it will if it is intended that the 50% of the network is to be allocated to an entire group to which this set of actors belong. In the latter case, not only does the enforcer need to assure that the overall network consumption restriction is enforced but some mediating entity must also implement an allocation strategy to manage sharing of the finite resource among individual members. For this reason, the actor ontology distinguishes between the concept of an *ActorGroup* (i.e., a kind of group that applies enforcement of a policy to the members individually) and a *GroupActor* (i.e., a kind of group that applies enforcement of a policy to the group as a whole).

The actions ontology defines various types of basic activities such as monitoring, communication, moving, accessing, and so forth. It also defines subclass relations between action classes, the *partOf* relation for composite actions and the *implementedBy* relation between abstract action classes and operations in the given software environment, thereby grounding the ontology concepts in specific platform capabilities. 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.[3]

For a given application, the core ontologies are 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.

3.2 KAoS Ontology Management

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,[4] additional application-specific or platform-specific ontologies then can be loaded dynamically from KPAT or programmatically using the appropriate Java method.



22-rdf-syntax-ns
::
Action.daml
Actor.daml
ActorClasses.daml
Entity.daml
EntityInstances.daml
Group.daml
GroupInstances.daml
KAoS
Message.daml
Place.daml
Policy.daml
PolicyConflict.daml
UltralogAction.daml
UltralogDefaultPolicy.daml
UltralogEntity.daml
UltralogGroup.daml
UltralogOntologies.daml
daml+oil
rdf-schema

Classes: |http://ontology.coginst.uwf.edu/Policy.daml#|::|NegAuthorizationPolicy|

Properties: |http://ontology.coginst.uwf.edu/Policy.daml#|::|hasDescription|

Instances Defined: |http://ontology.coginst.uwf.edu/Policy.daml#|::|AnySite|
http://ontology.coginst.uwf.edu/Policy.daml#	::	AnySite
http://ontology.coginst.uwf.edu/Policy.daml#	::	ActorSite
http://ontology.coginst.uwf.edu/Policy.daml#	::	TargetSite

Imported Namespaces: |http://www.daml.org/2001/03/daml+|::oil

Fig. 2. KPAT ontology namespace management window.

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.

[3] This distinction allows reasoning about actions on policies and the policy framework without resorting to the use of special “metapolicy” mechanisms.
[4] Pronounced “KAY-pat”

Figure 2 shows the KPAT ontology namespace window with a list of ontologies loaded from our DARPA Ultra*Log project work (*http://www.ultralog.net*). The right side of the window shows the concepts from the namespace that was just loaded, as divided into classes, properties, instances, and imported ontologies.

DAML files are intended to reside on the Web with namespaces defined by a URI. This creates a problem when the system has to be tested or deployed in an environment not connected to the Internet. We have solved this problem by gathering ontologies into a local directory and either serving them by a specialized Web proxy that will that will reroute requests to the stored files[5] or simply by reconfiguring an existing Web server such as Apache to simulate a list of URLs. Another solution is to use the persistence mechanism provided by some DAML tools (described later) to load the required ontologies from the Internet, store the persistence image, and then distribute the image with preloaded ontologies along with our KAoS code distribution.

3.3 Dynamic Creation of Ontology Concepts

Some concepts in the ontology have to be created dynamically, since they reflect the changing elements of the application environment. The DAML representing these dynamic elements must be generated either automatically or manually. An example of automatic creation is when new concepts are generated as the result of an agent registering to a KAoS domain. An example of manual creation of new concepts is when users explicitly create new policies using KPAT. In both these cases, KAoS employs the Jena Semantic Web Toolkit by HP Lab Bristol (*http://www.hpl.hp.com/semweb*) to incrementally build and finally output the completed DAML definition to the ontology repository managed by the Directory Service. We have had a very positive experience with Jena; generally, it provides a generic, comprehensive and reliable tool with which to build DAML.

Nevertheless, there are some minor issues with the Jena-generated DAML. One is that it contains some redundant information. For instance, if the concept being defined is an instance of some class, the DAML generated by Jena will naturally include information that the particular concept used to describe the type of the instance is a *daml:Class*. As this particular concept was previously obtained from the ontology repository to which the generated definition will now be committed, the redundant class information results in unnecessary computation. Our experience shows that the amount of the redundant information in a DAML output created in the described scenario consists of about 50 to 60 percent of all the definitions[6]. To remove this redundant information KAoS contains a trimming module for analyzing Jena output and removing all the DAML definitions referring through an *rdf:about* statement to the concept with a namespace already loaded into the KAoS ontology repository.

A second problem showed up for a DAML output from Jena containing a *daml:collection* construct. Jena converts such collections into *daml:List* constructs

[5] The Java Virtual Machine in which the Directory Service is running has to be provided the name of the host and the port number of the Web proxy serving the ontologies by using the Java environment properties (*proxyHost* and *proxyPort*).

[6] See *http://ontology.coginst.uwf.edu/Jena/Ontology%20Trimming.html* for an example.

and uses a non-unique identifier to refer to each element of the list (*A1*, *A2*, etc.). If, as happens frequently, more than one DAML definition contains such identifiers, the ontology repository becomes confused. Thus, the KAoS DAML trimmer adds a unique prefix to these identifiers.

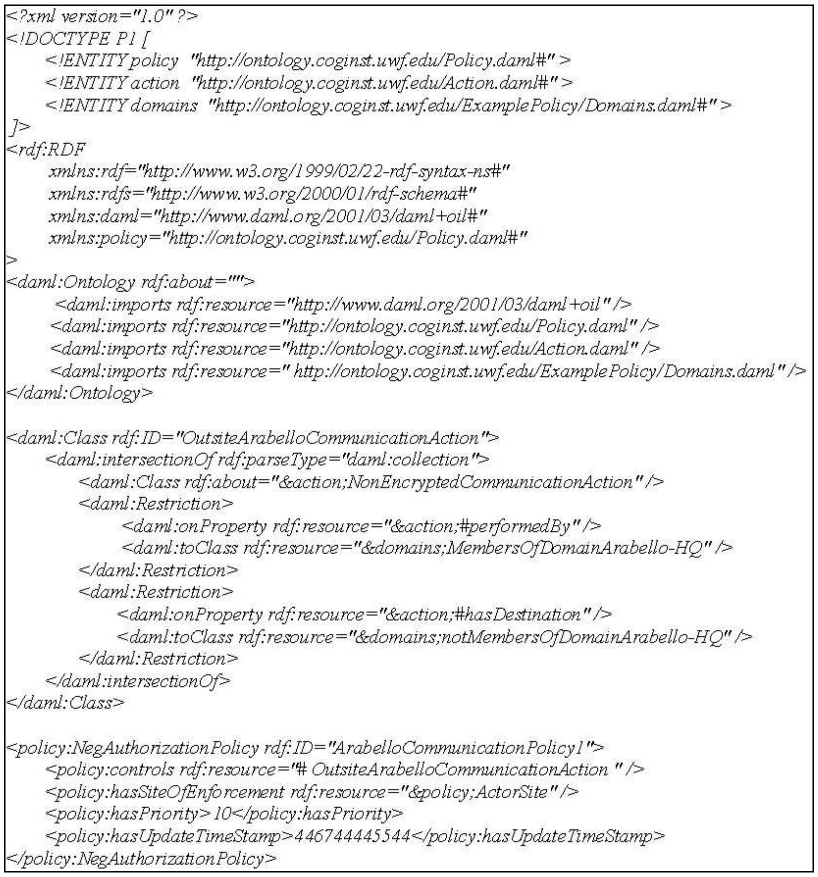
4 Policy Representation Considerations

In KAoS, policies express authorization or obligation constraints for some type of action performed by one or more actors in some situation [3]. 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 [6; *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.

Figure 3 below presents an example of a simple policy. The policy forbids any member of the Arabello domain from communicating with any entity outside the domain. In KAoS, a policy is represented as a DAML instance of one of the four types of policy (positive or negative authorization, 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. An additional set of properties is used to determine the actual meaning of the policy. The most important property value is the name of a controlled action class. Authorization policies use it to specify the action being authorized or forbidden. Instances of the obligation policy class contain additionally a trigger value that creates the obligation, which is also a name of the class of actions. Penalty properties contain a value that corresponds to a class of actions to be taken following a policy violation.

All these three properties mentioned above require use of action classes as instances within property values of a policy. It would be desirable to have a means to define a range of the policy properties restricting it to subclasses of a certain class, i.e., in this case the KAoS *Action* class. DAML does not currently support this feature, however this feature is included in the OWL Requirements document [8] as R13. The current workaround in DAML is to define the range as *http://www.daml.org/2001/03/daml+oil#Class*.

Typically any action classes required to support a new policy are generated automatically by KAoS when the policy is defined. 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.



```
<?xml version="1.0" ?>
<!DOCTYPE P1 [
     <!ENTITY policy  "http://ontology.coginst.uwf.edu/Policy.daml#" >
     <!ENTITY action  "http://ontology.coginst.uwf.edu/Action.daml#" >
     <!ENTITY domains  "http://ontology.coginst.uwf.edu/ExamplePolicy/Domains.daml#" >
]>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
     xmlns:policy="http://ontology.coginst.uwf.edu/Policy.daml#"
>
<daml:Ontology rdf:about="">
      <daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil" />
     <daml:imports rdf:resource="http://ontology.coginst.uwf.edu/Policy.daml" />
     <daml:imports rdf:resource="http://ontology.coginst.uwf.edu/Action.daml" />
     <daml:imports rdf:resource=" http://ontology.coginst.uwf.edu/ExamplePolicy/Domains.daml" />
</daml:Ontology>

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

Fig. 3 Example KAoS policy

5 Inference Engine Integration

Three inference engines were reviewed for use with KAoS: FaCT [7], DAMLJess [10], and the Java Theorem Prover (JTP) (*http://www.ksl.stanford.edu/software/JTP*). We were looking at three main criteria: 1. degree of full DAML support, 2. adequacy of the query interface, and 3. likelihood of good support and continued development of the tool. JTP seemed the best choice at the time, and was integrated into KAoS. One problem noted early on with JTP was the time required to assert new ontologies into the inferencing engine. However, the steady improvement of JTP has led to a dramatic increase in its performance, an order of magnitude or more in some cases. Currently, loading of the KAoS core ontologies takes less than 16 seconds on Pentium III 1.20 GHz with 640 MB RAM. Adding the definition of complexity similar to the policy presented on Figure 3 takes less than 340ms.

5.1 Generic Query Interface for DAML/RDF concepts

JTP provides a KIF (*http://cl.tamu.edu*) interface to perform queries. KAoS extends this capability using Java to provide a query interface specialized to certain kinds of relations in RDF graphs and in DAML ontologies. The basic features of this interface allow querying for:

- All the properties applicable to a given class;
- The range of some property for a certain class,
- All the known subclasses of a given class,
- All the known instances of the given class.

The complexity of these queries is considerable, since satisfying them requires examining a large portion of the ontologies, and traversing subproperty and subclass relations. For this reason, the results of these queries are cached. The cache is selectively flushed when a new ontology is loaded.[7]

Even though the query interface is used within a specialized editor its functionality can be used in a generic DAML editor. The editor would require these kinds of inferencing in order to present to a user a comprehensive view of the particular ontology concept, as this information can and usually is distributed throughout a DAML file or files.

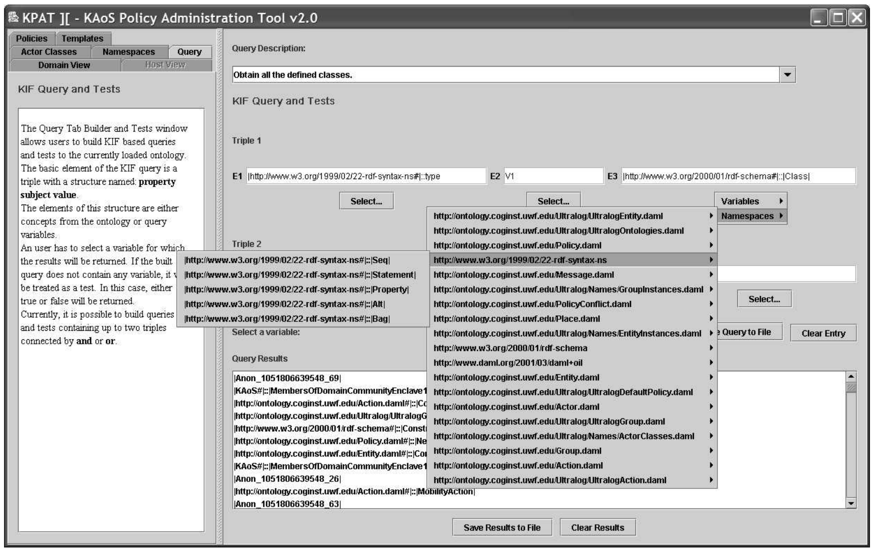
5.2 KIF Query Builder

The KIF Query Builder, developed and incorporated within KPAT and shown in Figure 4, allows users to manually build KIF queries for the currently loaded ontologies. Currently, it is possible to build queries containing up to two triples, of the form *(property subject value),* connected by *and* or *or*. The elements of this structure are either a concept from the ontology or query variables.

The user interface allows the user to build the query by selecting from among all the possible values of properties or concepts within a given namespace. A variable to which the results will be returned must also be selected; otherwise, the query will be treated as a yes/no query. In the latter case, it will return either true or false after evaluation.

The query builder can be used manually to put queries to the ontology repository. Equally important, it allows developers to test any queries necessary for platform-specific or application-specific extension to the system, save them to a file, and export them into Java code as strings used to initialize queries. This solves the problem of generating the ontology-based vocabulary for the Java code, necessary for interacting with the inference engine.

[7] The query interface is extensively used within KPAT; an example of its use is presented in section 6.2.


KPAT][- KAoS Policy Administration Tool v2.0
Policies
Templates
Actor Classes
Namespaces
Query
Domain View
Host View
Select...
Select...
Variables
Namespaces
http://ontology.coginst.uwf.edu/Ultralog/UltralogEntity.daml
http://ontology.coginst.uwf.edu/Ultralog/UltralogOntologies.daml
http://ontology.coginst.uwf.edu/Policy.daml
http://www.w3.org/1999/02/22-rdf-syntax-ns
http://ontology.coginst.uwf.edu/Message.daml
http://ontology.coginst.uwf.edu/Ultralog/Names/GroupInstances.daml
http://ontology.coginst.uwf.edu/PolicyConflict.daml
http://ontology.coginst.uwf.edu/Place.daml
http://ontology.coginst.uwf.edu/Ultralog/Names/EntityInstances.daml
http://www.w3.org/2000/01/rdf-schema
http://www.daml.org/2001/03/daml+oil
http://ontology.coginst.uwf.edu/Entity.daml
http://ontology.coginst.uwf.edu/Ultralog/UltralogDefaultPolicy.daml
http://ontology.coginst.uwf.edu/Actor.daml
http://ontology.coginst.uwf.edu/Ultralog/UltralogGroup.daml
http://ontology.coginst.uwf.edu/Ultralog/Names/ActorClasses.daml
http://ontology.coginst.uwf.edu/Group.daml
http://ontology.coginst.uwf.edu/Action.daml
http://ontology.coginst.uwf.edu/Ultralog/UltralogAction.daml
http://www.w3.org/1999/02/22-rdf-syntax-ns#	::	Seq
http://www.w3.org/1999/02/22-rdf-syntax-ns#	::	Statement
http://www.w3.org/1999/02/22-rdf-syntax-ns#	::	Property
http://www.w3.org/1999/02/22-rdf-syntax-ns#	::	Alt
http://www.w3.org/1999/02/22-rdf-syntax-ns#	::	Bag
Select...
Query to File
Clear Entry
Save Results to File
Clear Results

KIF Query and Tests

The Query Tab Builder and Tests window allows users to build KIF based queries and tests to the currently loaded ontology. The basic element of the KIF query is a triple with a structure named: **property subject value**.
The elements of this structure are either concepts from the ontology or query variables.
An user has to select a variable for which the results will be returned. If the built query does not contain any variable, it w be treated as a test. In this case, either true or false will be returned.
Currently, it is possible to build queries and tests containing up to two triples connected by **and** or **or**.

Query Description: Obtain all the defined classes.

KIF Query and Tests

Triple 1

E1 |http://www.w3.org/1999/02/22-rdf-syntax-ns#|::type E2 V1 E3 |http://www.w3.org/2000/01/rdf-schema#|::|Class|

Triple 2

Select a variable:

Query Results

```
|Anon_1051806639548_69|
|KAoS#|::|MembersOfDomainCommunityEnclave1
|http://ontology.coginst.uwf.edu/Action.daml#|::|Co
|http://ontology.coginst.uwf.edu/Ultralog/UltralogG
|http://www.w3.org/2000/01/rdf-schema#|::|Const
|http://ontology.coginst.uwf.edu/Policy.daml#|::|Ne
|http://ontology.coginst.uwf.edu/Entity.daml#|::|Co
|KAoS#|::|MembersOfDomainCommunityEnclave1
|Anon_1051806639548_26|
|http://ontology.coginst.uwf.edu/Action.daml#|::|MobilityAction|
|Anon_1051806639548_63|
```

Fig. 4. KPAT KIF Query Builder

5.3 Analyzing Policies

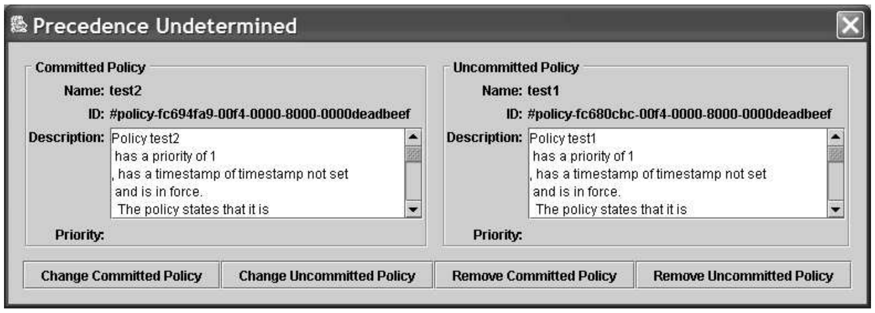
Some of the most important features of description-logic-based policy representation and reasoning show their advantages as part of policy analysis. Among others, these include subsumption-based reasoning, determination of disjointness, and instance classification [1]. The first two features are used mainly during the kinds of analysis associated with policy administration. Instance classification is especially valuable for policy exploration, disclosure, and distribution—it is used, for instance, to determine which entities belong to a given domain or if a resource being accessed by a given action is within a range constrained by policy. Sections 5.3.1 and 5.3.2 discuss the problems of policy administration and policy exploration and disclosure in greater detail. Most of the capabilities discussed in these sections are already available in KAoS; a few are currently under development. The diversity of applications of KAoS will no doubt continue to fuel even more sophisticated forms of analysis in the future [2, 3, 14].

5.3.1 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,[8] and, if desired, facilitating any necessary modifications.

[8] Visualization features are under development.

One important type of interaction is a policy conflict [3, 14]. 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.[9] If precedence cannot be determined otherwise, KAoS will ask the administrator to determine the appropriate action (Figure 5).


Precedence Undetermined

Committed Policy
Name: test2
ID: #policy-fc694fa9-00f4-0000-8000-0000deadbeef
Description: Policy test2
has a priority of 1
, has a timestamp of timestamp not set
and is in force.
The policy states that it is
Priority:

Uncommitted Policy
Name: test1
ID: #policy-fc680cbc-00f4-0000-8000-0000deadbeef
Description: Policy test1
has a priority of 1
, has a timestamp of timestamp not set
and is in force.
The policy states that it is
Priority:

Fig. 5. Notification about policy conflict and options available to the administrator.

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;[10]
- *Harmonize Policy*: the controlled action of the selected overlapping policy can be modified using an automatic harmonization algorithm to eliminate their overlap; see [14] for details.. This required modification of the restrictions in of the policy controlled actions by building either intersection (by using *daml:intersectionOf*) or differences (by using *daml: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

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

[10] 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, which has the largest or smallest scope, whether negative or positive authorization trumps by default, whether subdomains takes precedence over superdomains or vice versa, and so forth.

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 unsatisfiable 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.

5.3.2 Policy Exploration and Disclosure

A human user or software component may want to use KPAT or a software interface to answer "what if" questions (about the consequences of policy modifications) or "how to" questions (to help them understand how to satisfy existing policies in performing some action).[11] In general, the answers to these queries are decided by inferring whether some concrete action falls into a category of action controlled 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 providing 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.
- *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.[12]
- *Make Compliant:* action an actor tries to perform by informing it about the required changes to the action based on the existing policies. For instance it

[11] Note that policy restrictions may disallow certain types of information to be disclosed in response to such queries.

[12] Admittedly, a sophisticated form of this type of analysis would start to resemble a planning problem in some respects. We are exploring the relationship between policy management and planning technologies in new research with James Allen.

can try to send a message about particular subject to a few actors and the list of actors is restricted to some subset or maybe the list has to be extended for some required recipients; maybe the content of the actor message has to be transformed by striping off some information.

- *Get Consequences:* of some action by observing/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 investigated by us.

5.4 Persistence and Transaction Support

Support for persistence and transactions is essential for any system with ambitions for use in a production environment. Our work on the use of policy to enhance system survivability within the DARPA Ultra*Log program, for example, has extremely demanding requirements for security and reliability. Although the state of inference mechanisms are not directly accessible to outside methods, due to their being embedded in the Directory Service, JTP provides the necessary foundation to build the required mechanisms.

For example, JTP's snapshot method allows for fast checkpointing of the state and fast recovery. This mechanism is used in KAoS during every commit of a new ontology. Any inconsistency will result in an exception with a JTP-provided explanation, and the pre-commit state will be recovered.

Additionally, JTP allows saving its current state to a file. This process is slower than checkpointing, however if this feature is used conservatively and in conjunction with the recording of assertions between file saves it can be used for recovery and provides persistence.

5.5 Issues in Handling Dynamism

Because relevant aspects of the KAoS ontologies need to stay consistent with rapidly changing state of application entities being governed by policy, a variety of information about instances is stored in the inferencing engine context. Without this information, it would be impossible to do reasoning about instance classification. When application entities represented as instances are removed or changed in certain ways, the inference engine needs a way to remove previously asserted information.

JTP provides an *untell* mechanism for this purpose. It relies on the checkpoint mechanism to roll back to the state before the given assertion was made and then rolls forward again by reasserting all subsequent statements except for the one removed. This implementation of untell, while useful as a last resort, is too slow to work practically in applications such as ours. Stanford is considering the development of an alternative implementation.

As a necessary workaround, KAoS currently keeps information about the removed instances in a table outside of the main JTP context and uses this information to filter information returned by JTP. Information about any properties whose values may change is kept in our own implementation of a separate classifier instead of in JTP.

This classifier can be implemented either using a distinct context of JTP, a simple table or rely on the existing classifying service. The information about properties is used to classify instances and combined with the results of instance classification returned from JTP.

6 Ontology-driven System Architecture

In this section we consider the benefits and problems of using ontologies as a central aspect of system design. An ontology allows for great flexibility in design and deployment, however careful attention to performance-sensitive aspects of the system is essential. Additional problems arise at two boundaries: where the reasoning system meets the human world and where it meets the systems being governed by policy. Our approach to addressing these issues is described in this section.

6.1 Ontology as Configuration Glue

The framework nature of KAoS means that the installation configuration can vary. Since the role of each software component is related to concepts defined in specialized ontologies it is relatively easy to associate these components (enforcers, classifiers, policy editors, etc.) with an appropriate ontology definition. Such mappings are registered in proper software factories, creating a new Java component on demand (see Figure 1). KAoS always checks if particular factory consists of a specialized component for handling the given ontology concept and if so, uses it instead of the generic functionality.

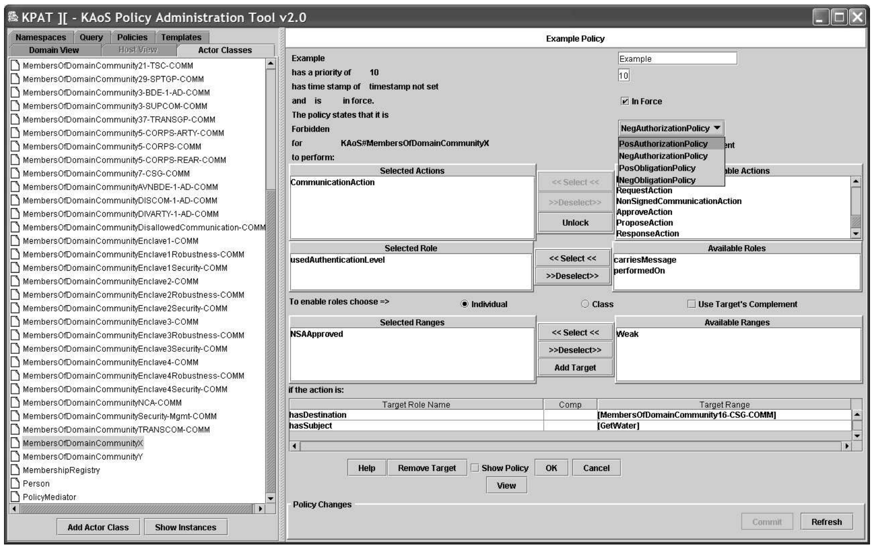
6.2 Translating Ontology into Notions of the Legacy System

When policy leaves the Directory Service it typically has to translate from DAML into some format, which is compatible with the integrated legacy systems.

KAoS communicates to the outside world DAML originate information using a concept of map relating ontology properties to the name of the class defining its range as well to the list with cached instances of that class when the map left the Directory Service. A particular system can use the given cached instance for its computation; also in any moment it can refresh them by contacting the Directory Service and providing the name of the range.

6.3 Graphical Interface to Ontology Concepts

The KPAT graphical interface hides the complexity of the DAML representation from users. On the other hand, its unique user experience is achieved through the use of ontology. The user is always presented with a complete set of choices, which are valid in the given context.



MembersOfDomainCommunity21-TSC-COMM
MembersOfDomainCommunity29-SPTGP-COMM
MembersOfDomainCommunity3-BDE-1-AD-COMM
MembersOfDomainCommunity3-SUPCOM-COMM
MembersOfDomainCommunity37-TRANSGP-COMM
MembersOfDomainCommunity5-CORPS-ARTY-COMM
MembersOfDomainCommunity5-CORPS-COMM
MembersOfDomainCommunity5-CORPS-REAR-COMM
MembersOfDomainCommunity7-CSG-COMM
MembersOfDomainCommunityAVNBDE-1-AD-COMM
MembersOfDomainCommunityDISCOM-1-AD-COMM
MembersOfDomainCommunityDIVARTY-1-AD-COMM
MembersOfDomainCommunityDisallowedCommunication-COMM
MembersOfDomainCommunityEnclave1-COMM
MembersOfDomainCommunityEnclave1Robustness-COMM
MembersOfDomainCommunityEnclave1Security-COMM
MembersOfDomainCommunityEnclave2-COMM
MembersOfDomainCommunityEnclave2Robustness-COMM
MembersOfDomainCommunityEnclave2Security-COMM
MembersOfDomainCommunityEnclave3-COMM
MembersOfDomainCommunityEnclave3Robustness-COMM
MembersOfDomainCommunityEnclave3Security-COMM
MembersOfDomainCommunityEnclave4-COMM
MembersOfDomainCommunityEnclave4Robustness-COMM
MembersOfDomainCommunityEnclave4Security-COMM
MembersOfDomainCommunityNCA-COMM
MembersOfDomainCommunitySecurity-Mgmt-COMM
MembersOfDomainCommunityTRANSCOM-COMM
MembersOfDomainCommunityX
MembersOfDomainCommunityY
MembershipRegistry
Person
PolicyMediator

Example
has a priority of 10
has time stamp of timestamp not set
and is in force.
The policy states that it is
Forbidden
for KAoS#MembersOfDomainCommunityX
to perform:

Example
10
✓ In Force
NegAuthorizationPolicy
PosAuthorizationPolicy
NegAuthorizationPolicy
PosObligationPolicy
NegObligationPolicy

Selected Actions: CommunicationAction
Available Actions: RequestAction, NonSignedCommunicationAction, ApproveAction, ProposeAction, ResponseAction

Selected Role: usedAuthenticationLevel
Available Roles: carriesMessage, performedOn

To enable roles choose => [✓ Individual] Class ☐ Use Target's Complement

Selected Ranges: NSAApproved
Available Ranges: Weak

if the action is:

Target Role Name	Comp	Target Range
hasDestination		[MembersOfDomainCommunity16-CSG-COMM]
hasSubject		[GetWater]

Fig. 6. KPAT generic policy builder – an example of ontology guided interface

As in the case of the generic policy editor shown on Figure 6, 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 action to actors by the *performedBy* property. When the user selects a particular action type again 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.

7 Conclusion

We have shown that the use of description logic provides significant advantages in the design and development of a complex software system. Although some problems arose from the expressive limitations of DAML, we were able to find effective workarounds in practice, and the performance of available DAML technology has improved significantly during the course of this project. We believe that the techniques we have developed for using DAML in an agent-based application are of general utility and can be re-used in other systems. This work provides practical evidence in support of the thesis that the use of ontologies as a central paradigm in an object-oriented programming scenario is an effective design strategy.

Acknowledgments
The authors gratefully acknowledge the sponsorship of this research by the NASA Cross-Enterprise and Intelligent Systems Programs, and a joint NASA-DARPA ITAC

grant. Additional support was provided by DARPA’s CoABS, Ultra*Log, and CoSAR-TS programs. We are also grateful for the contributions of Austin Tate, Niranjan Suri, Paul Feltovich, Richard Fikes, Jessica Jenkins, Rich Feiertag, Timothy Redmond, Sue Rho, Ken Ford, Mark Greaves, Jack Hansen, James Allen, and Robert Hoffman.

References

[1] Baader, F., Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P. (Ed.) (2003). The Description Logic Handbook. Cambridge University Press,

[2] 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. Kluwer, in press.

[3] Bradshaw, J. M., Beautement, P., Breedy, M. R., Bunch, L., Drakunov, S. V., Feltovich, P., Hoffman, R. R., Jeffers, R., Johnson, M., Kulkarni, S., Lott, J., Raj, A. K., Suri, N., & Uszok, A. (2003). Making agents acceptable to people. In N. Zhong and J. Liu (Eds.), Handbook of Intelligent Information Technology. Amsterdam: IOS Press, in press.

[4] Bradshaw, J. M., Boy, G., Durfee, E., Gruninger, M., Hexmoor, H., Suri, N., Tambe, M., Uschold, M., & Vitek, J. (Ed.). (2003). *Software Agents for the Warfighter. ITAC Consortium Report.* Cambridge, MA: AAAI Press/The MIT Press, in preparation.

[5] Bradshaw, J. M., Dutfield, S., Benoit, P., & Woolley, J. D. (1997). KAoS: Toward an industrial-strength generic agent architecture. In J. M. Bradshaw (Ed.), Software Agents. (pp. 375-418). Cambridge, MA: AAAI Press/The MIT Press.

[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, London,

[7] Horrocks, I., Sattler, U., Tessaris, S. and Tobies., S. (2000). How to decide query containment under constraints using adescription logic. In Proceedings of LPAR'2000,

[8] Heflin, J. (2003). Web Ontology Language (OWL) Use Cases and Requirements. http://www.w3.org/TR/webont-req/,

[9] 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. ACM Press,

[10] Kopena, J. (2002) DAMLJess web site.
http://edge.mcs.drexel.edu/assemblies/software/damljesskb/damljesskb.html

[11] Paolucci, M., Kawamura, T., Payne, T. and Sycara, K. (2002). Semantic Matching of Web Services Capabilities. In Proceedings of the 1st International Semantic Web Conference (ISWC).

[12] Sierhuis, M. (2002). Brahms - Modeling and Simulating Work Practice. Univ. of Amsterdam Press,

[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] Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., Hayes, P., Breedy, M., Bunch, L., Johnson, M., Kulkarni, S. and Lott, J. (2003). KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction and Enforcement. In Proceedings of IEEE Workshop on Policy 2003.