DAML-based Policy Enforcement for Semantic Data Transformation and Filtering in Multi-agent Systems

Niranjan Suri, Jeffrey Bradshaw, Mark Burstein, Andrzej Uszok, Brett Benyo, Maggie Breedy, Marco Carvalho, David Diller, Renia Jeffers, Matt Johnson, Shri Kulkarni, James Lott

Institute for Human & Machine Cognition
University of West Florida
40 S. Alcaniz St., Pensacola, FL 32501
++1-850-202-4462

{nsuri,jbradshaw,auszok,mbreedy,
mcarvalho,rjeffers,mjohnson,
skulkarni,jlott}@ai.uwf.edu

BBN Technologies
10 Moulton St., Cambridge, MA 02138
++1-617-873-3861

{burstein,bbenyo,ddiller}@bbn.com

ABSTRACT

This paper describes an approach to runtime policy-based control over information exchange that allows a far more fine-grained control of these dynamically discovered agent interactions. The DARPA Agent Markup Language (DAML) is used to represent policies that may either filter messages based on their semantic content or transform the messages to make them suitable to be released. Policy definition, management, and enforcement are realized as part of the KAoS architecture. The solutions presented have been tested in the Coalition Agents Experiment (CoAX) - an experiment involving coalition military operations.

Keywords

Policies, Policy-based Control, Information Release Policies, Semantic Filtering, Data Transformation Policies, Coalition Operations, CoAX, CoABS Grid, DAML, DAML-S, KAoS, Nomads.

1. INTRODUCTION

Software agents have been proposed as a solution to integrate existing heterogeneous and stovepiped information systems. Agent and service description mechanisms, discovery and matchmaking services, and agent communication languages all contribute to agents achieving ad-hoc and dynamic interoperability at runtime as opposed to at design time. Mechanisms such as DAML services and the CoABS Grid make it possible for agents to dynamically interoperate thereby increasing their effectiveness. These capabilities extend agent autonomy to encompass communication and information exchange. However, in sensitive or critical application areas such as the military or competitive business environments involving proprietary information this kind of transparent interoperability raises concerns about the nature of the information being exchanged. The dynamic and autonomous nature of this process complicates the task of system designers who wish to maintain control over the flow of information to and from their agents.

This paper describes an approach to provide runtime policy-based control over information exchange. Two different control mechanisms are discussed: semantic (content-based) filtering of messages as well as in-stream transformation of messages. Both of these control mechanisms are driven by policies at run-time. These mechanisms allow a far more fine-grained control over dynamic and autonomous agent interactions than a suite of policies created at design time, before all possible interactions are known. With such an approach, we hope to increase the confidence with which system designers will adopt agent-based approaches to building dynamic, heterogeneous systems.

Central to our approach is the KAoS architecture which provides a set of services for policy-based control over agent behavior. KAoS provides a framework for policy definition, management, distribution, and enforcement. The capabilities described in this paper have been realized as extensions within the KAoS framework which uses the DARPA Agent Markup Language (DAML) is used to represent policies. Our current implementation operates on top of the DARPA Control of Agent-based Systems (CoABS) Grid, which provides agent registration, lookup, and messaging services. However, the solution is portable to other platforms that provide the same basic services. Finally, the Nomads mobile agent system and the Flexible Data Feeds Framework (FlexFeed) are used to realize the enforcement mechanisms.

The rest of the paper is organized as follows. Section two provides an overview of the CoABS Grid, DAML, KAoS, Nomads, and FlexFeed components. Section three describes the Coalition Agents Experiment (CoAX) scenario that motivated the development of these capabilities. Section four presents the architecture of KAoS. Section five describes the implementation of the semantic filtering capability. Section six describes the implementation of the dynamic transformation capability. Finally, section seven concludes with a summary and discusses future work.

2. COMPONENTS OVERVIEW

2.1 CoABS Grid

The DARPA CoABS Grid is an integration platform designed to support runtime interoperability of software agent systems [9,12]. The CoABS Grid was developed under the auspices of the

DARPA Control of Agent-based Systems program. It is designed to be an agent-friendly layer on top of the Jini framework [14]. The CoABS Grid provides, among other capabilities, agent registration, lookup, and messaging.

Agent registration is handled by placing a descriptor for an agent along with a proxy for that agent into the Jini Lookup Service (LUS). Agent lookup is achieved by passing a partial descriptor to the Grid which then returns a set of proxies for the matching agents. Once a proxy to an agent has been retrieved, a message may be sent to the agent by invoking a method on the target agent's proxy.

The Grid also provides a logging service to keep track of the message traffic and a GUI administration tool to help configure and startup the Grid services, visualize registered agents, and perform other kinds of monitoring operations.

2.2 DAML

The DARPA Agent Markup Language (DAML) is a semantic representation language jointly developed by the DARPA DAML program and the World Wide Web Consortium (W3C). Built on top of RDF, a description language developed by W3C and its contributing members, DAML provides a capability to describe classes, properties, and descriptions built using those in a syntax based on XML and SGML. A key notion in RDF and DAML is that there is a universal namespace built on URI's, so that ontologies developed at different locations and published at different locations on the web can refer to each other. DAML enhances RDF by providing a formal semantics, and includes additional constructs for indicating such things as the equivalence or disjointness of different classes, key concepts required to do useful classification inferences. Within the DAML program, a group of researchers has developed DAML-S [2,3], an ontology for describing web and agent services. This ontology, consisting of separate parts for service profiles (advertising services), service process models (including semantic descriptions of what are essentially messaging APIs), and service groundings, that describe how atomic processes are to be mapped into various messaging formats such as provided by SOAP, HTML, the CoABS Grid, and so forth. We make extensive use of both DAML and DAML-S in the examples described in this paper.

2.3 KAoS

KAoS is a collection of componentized agent services compatible with several popular agent frameworks, including Nomads [15,16], the DARPA CoABS Grid [9,12], the DARPA ALP/Ultra*Log Cougaar framework (http://www.cougaar.net), CORBA (http://www.omg.org), and Voyager (http://www.recursionsw.com/osi.asp). 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). For a full description of KAoS, the reader is referred to [5,6,7].

There are two key KAoS services relevant to the effort described here: *policy services* and *domain services.*

Policy services are used to define, manage, and enforce constraints assuring coherent, safe, effective, and natural interaction among teams of humans and agents. Knowledge is represented declaratively in DAML ontologies. The current version of the KAoS Policy Ontologies (KPO) defines basic ontologies for groups, actors, actions, places, messages, various entities related to actions (e.g., computing resources), and policies. We have extended these ontologies to represent simple atomic Java permissions, as well as more complex Nomads, and KAoS policy constructs. New extensions to the ontologies are continually being developed to represent domain- and application-specific information.

Domain services are used to facilitate the structuring of agents into complex organizational structures, administrative groups, and dynamic task-oriented teams that may span many hosts, platforms, and locations. Domains also provide a common point of administration and policy enforcement. Through various DAML property restrictions, a given policy can be variously scoped, for example, either to individual agents, to agents of a given class, to agents belonging to intensionally- or extensionally-defined domains or teams, or to agents running in a given physical place or computational environment (e.g., host, VM).

2.4 Nomads

Nomads is a mobile agent system for Java-based agents. Nomads provides two implementations: Oasis and Spring. Oasis incorporates a custom Java-compatible Virtual Machine (named Aroma) whereas Spring is a pure Java implementation. The Aroma VM is a clean-room VM designed to provide the enhanced capabilities of execution state capture and resource control [15,16]. Building on top of the capabilities of Aroma, Oasis provides three capabilities:

- strong mobility where agents can move while preserving their execution state
- forced mobility where, completely transparent to them, agents may be moved from one system to another by an external asynchronous request
- secure execution of untrusted agents while protecting the host from denial of service and other forms of attack

The Spring implementation is fully interoperable with Oasis but does not provide the above features. Spring is well-suited for lightweight applications or environments that do not support Aroma (e.g., mobile telephones or PocketPC).

In the context of this paper, the Nomads system is used to provide policy enforcement mechanisms and in particular the enforcement of the dynamic data transformation policies. The FlexFeed architecture (described below) builds on top of Nomads and uses Nomads to deploy mobile agents to act as relay and transformation nodes within a data path.

2.5 FlexFeed

The FlexFeed framework [8] is a middleware communications service for multi-agent systems. FlexFeed provides bandwidth-efficient communication while at the same time addressing the limitations of processing capabilities on nodes in the communications framework. In particular, FlexFeed is designed to support low-powered sources and sinks while processing, relaying, and transforming data in-stream on intermediate nodes. FlexFeed is opportunistic in taking advantage of available

resources and can adapt to a changing environment through mobility.

At the core of FlexFeed are a set of services and interfaces that facilitate the process by which agents can request and provide data streams. Data streams are transferred between agents as a sequence of individual messages. In general, data can be generated by one or more agents (sources agents) and possibly aggregated and distributed to one or more sink agents. The role of the FlexFeed framework in this case is to continually evaluate data stream requests against communication and information release policies to establish and maintain logical distribution channels for the data streams.

The framework provides a simple API for service registration and request. The main difference is that the provisioning of the request will be checked and potentially modified, transparently to the agents, to enforce information release policies between two (or more) agents.

When an agent requests a data feed from some specific source, the framework calculates and establishes the data distribution path between the two agents. This can be a direct path between the two or, if required by policies, it can also involve relay agents through which the data will be filtered or modifed to comply with information release restrictions.

If data must be transformed between the source and destination agents to satisfy policies, the framework will deploy the necessary processing capabilities (usually in the form of a policy enforcement agent) to account for that. If the more than one agent is receiving the data, appropriate transformation agents are deployed along the path to ensure both policy enforcement and efficient bandwidth consumption with load distribution.

Figure 1 illustrates a simple example where two sink agents are receiving a data stream from the same source agent (a sensor). To satisfy local policies between the sensor and sink agent B, a transformation agent T has been deployed to participate in the data distribution path.

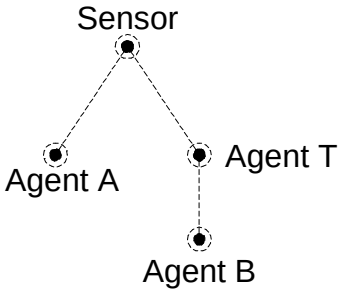

Figure 1: FlexFeed Data Distribution Path To Enforce Data Transformation

One important aspect to note here is that the deployment of the transformation agent T is transparent to the agent B. After requesting the feed, the agent B simply starts to receive the data stream but, for example, in lower resolution due to the policy restrictions.

There are currently two main implementations of the FlexFeed framework, one that ensures that every data packet is sent though the KAoS framework so each message is explicitly checked and evaluated against policies. A second implementation of the framework allows direct communication between the agents but only through the pre-defined data distribution path established by the framework, in compliance with the communication and information release policies provided by KAoS.

3. CoAX Binni Scenario

The Coalition Agents Experiment (CoAX) relies upon an unclassified fictitious military scenario named Binni [13] that was created to experiment with coalition military operations. Binni is set in the year 2012 and involves three imaginary countries in Africa – Binni, Gao, and Agadez. Due to a conflict in the region between these three countries, a multinational UN peacekeeping force is brought in stop the conflict. The multinational force includes the United States, the United Kingdom, Canada, and Australia. During the course of the scenario, a fourth imaginary country – Arabello – is called upon to join the coalition. The Binni scenario provides a rich and militarily-relevant setting for experimenting with agent-based systems for coalition operations.

One of the critical concerns in any coalition operation (military or civilian) involves protection of sensitive or proprietary information. The Binni scenario models the complexities and nuances of the relationships between different countries that make up the coalition peacekeeping force. For example, the US, UK, and Australia have a high degree of mutual trust whereas Gao, which is also a member of the coalition, is trusted to a lesser extent. Therefore, from the perspective of one country (such as the US), there are three different scopes for information sharing: agents that are part of the US, agents that are part of the UK and Australia, and agents that are part of Gao.

Moreover, Gao starts out as a member of the coalition force but is then found to be providing misinformation to the rest of the coalition in order to advance its own private agenda. When this deception is discovered, the trust relationships are altered and the degree of information sharing between agents has to be adapted accordingly.

Finally, during the scenario Arabello joins the coalition and new trust relationships must be established between the existing coalition members and Arabello. The coalition needs to be able to effectively manage their concerns about information released by its agents to Arabello's agents and vice versa.

4. KAoS POLICY ARCHITECTURE

KAoS provides a flexible policy architecture, which was used to implement the semantic filtering as well as the dynamic transformation policies. Figure 2 shows the overall KAoS architecture. From the perspective of this paper, the capabilities of KAoS may be divided into three categories: policy definition and editing, policy storage, propagation, and mapping, and policy enforcement. The major components of KAoS are:

- KPAT:[1] The KAoS Policy Administration Tool is used to specify, modify, browse, commit, and otherwise administer domains and policies. KPAT comes with a generic DAML policy editor; customized editors for

[1] Pronounced "KAY-pat."

different types of policies can also be created and accessed through a pop-up menu.

- Domain Manager: Domain Managers manage domain membership and are responsible for maintaining policy consistency. They store policies in the directory and distribute policies to Guards as appropriate.
- Directory Service: The Directory Service acts as a secure repository for policies. It can respond to a variety of queries from the domain manager and other trusted entities.
- Guards: The guards receive policies from the domain manager and enforce them with appropriate mechanisms. The Domain Manager maintains a mapping of guards and the policies for which they and the enforcers they manage are responsible.
- Enforcers: Components that are capable of enforcing particular types of policy on agents. New types of enforcers may be added based on the capabilities of the underlying execution and agent platforms.

A more detailed description of KAoS Policy and Domain Services can be found in [17].

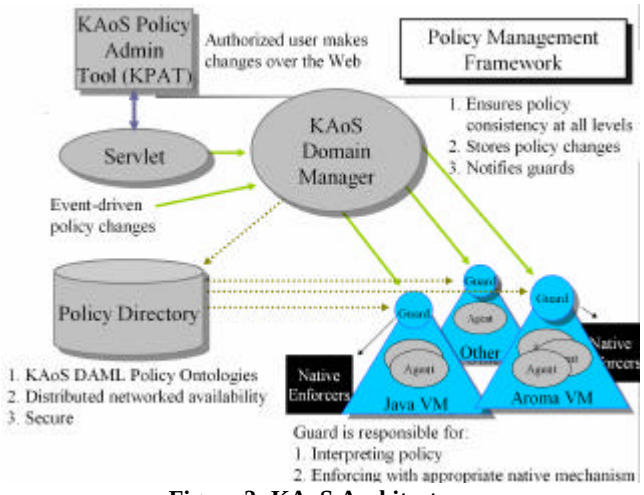

Figure 2: KAoS Architecture

5. SEMANTIC FILTERING BASED ON MESSAGE TYPES

In some situations, restricting agent communication based solely on attributes of the sending or receiving agent may be a too coarse grained form of control. Often, the content of the communication contains the critical features for determining if and how to filter the communication. For example, one might wish to restrict the sharing of sensitive personal data or proprietary business information. Just as we have used DAML-S as the language to express the capabilities and services of agents, we use DAML to express the semantic content of the information exchanged through services. Using DAML, we present a system for specifying and enforcing a semantic content filter. This system requires no modification of source code, allowing content filters to be dynamically defined during run-time.

We have demonstrated this system as part of the CoAX technology integration experiment. In the CoAX scenario, the country of Arabello joins the coalition. Consequently, a number of new agents and agent services need to be dynamically made available to coalition agents. These agents and services are dynamically discovered, resulting in a number of new agent interactions. For example, one interaction involves a coalition agent tasked to locate a hostile submarine and an Arabello agent capable of providing sensor reports from an underwater sensor grid. As new coalition partners, Arabello system administrators dynamically allow sensor contact reports to be sent to the coalition agent, but for security reasons, restrict the range of messages that could be sent outside of the Arabello domain. The limitation, described as part of a policy represented in DAML, limits these outgoing messages to those whose content are reports about a specific class of submarine, belonging to the enemy forces, but disallowing reports on other ships, such as those of Arabello itself.

5.1 Filter Specification

In order to enable domain administrators to specify a message content filter, it first becomes necessary to access the DAML ontology describing the possible outgoing message contents. This information is available as part of the DAML-S service description for each agent: specifically, the output message property in the agent's service process model. A DAML-S process model describes, among other things, the DAML classes representing the types of each service's inputs and outputs. The approach to specification of message filters we adopted enables a KAoS domain system administrator to specify or define a subclass of the most general allowed class of input or output messages that will be permitted to be sent or received by some class of agents.

For each content filter, a GUI is dynamically generated that lets the system administrator build up a specialized class definition that will be used as a filter by comparing it to each message being sent between the classes of agents covered by the policy. If the message is subsumed by this class, then the message is permitted to be sent or received in the case of a *positive authorization* policy, or blocked in the case of a *negative authorization* policy. The specialized DAML class is created by placing additional property range restrictions on the properties of the DAML class describing the message content specified in general by the agent's service process model. For each property to be restricted, we can create two types of DAML restrictions. The first, called a *toClass* restriction, requires that the value of the property be a member of a certain DAML class, the second, called a *hasValue* restriction, requires the property to have a specific value.

KPAT, our policy administration tool, provides an interface by which a system administrator can specify policies for interactions based on the properties of services and agents discovered dynamically at run-time. For the content filtering policies, the properties of the DAML class representing the output of each service are shown to the administrator in a dynamically generated GUI, along with a *toClass* restriction editor, and a *hasValue* restriction editor. The *toClass* editor contains a list of previously-defined classes that could be selected in order to further restrict the property's range. This list is developed by expanding and linearizing the subclass tree of the DAML class defining the original *daml:range* of the property for the message

class specified for the service. The *hasValue* editor allows the entry of freeform text representing the value of the property, or selecting from known instances of the range class, if it consists of a pre-defined closed list. Special graphical editors for certain DAML classes (e.g., dates and latitude/longitude coordinates) are also provided. In addition, if the *daml:range* is a *daml:Class*, meaning that the range of values of the property is a set of DAML class URIs, the possible values are automatically generated, just as for the *toClass* restriction.

The interface generated for our CoAX example is shown in Figure 3. By selecting *DieselSubmarine* in the *hasValue* editor for the property *objectClassification*, the system administrator is restricting outgoing *sensorReportResult* messages from the Arabello agent that is the subject of the policy to those with property *objectClassification* having the value *DieselSubmarine.*

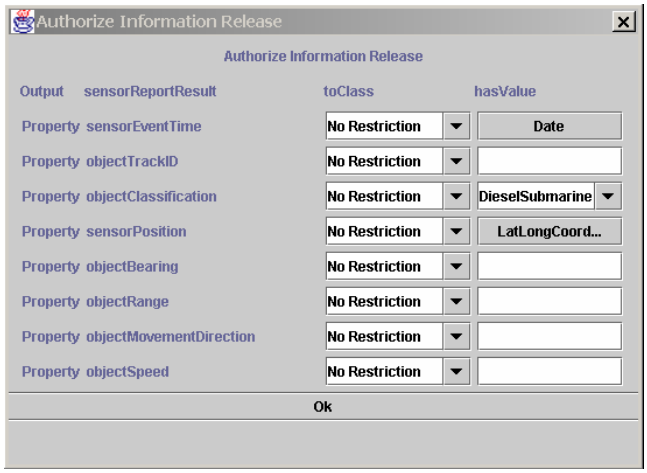

Figure 3: Semantic Content Filter Specification Dialog

5.2 Filter Generation

Once the classes describing the message properties that indicate which messages are to be filtered is specified through this GUI, a persistent DAML class is created, representing the full set of these restrictions. This new DAML class is defined as an intersection of the original output message class and a *class expression* (specified using **daml:Restriction**) for each property that is further restricted by a *toClass* or *hasValue* expression. In our example, the resulting DAML class is given below:

```
<daml:Class rdf:ID="RestrictedASWSensorReport">
 <daml:intersectionOf rdf:parseType="daml:collection">
  <daml:Class rdf:about="&asw;#ASWContactReport"/>
  <daml:Restriction rdf:ID>
   <daml:onProperty rdf:resource="&asw;#objectClassification"/>
   <daml:toClass rdf:resource="&vehicles;#DieselSubmarine"/>
  </daml:Restriction>
 </daml:intersectionOf>
</daml:Class>
```

5.3 Policy Conflict Resolution

Changes or additions to policies in force, or a change in status of an actor (e.g., an agent joining a new domain or moving to a new host) or some other entity require logical inference to determine first of all which policies are in conflict and second how to resolve these conflicts. We have implemented a general-purpose algorithm within KAoS for policy conflict detection and harmonization whose initial results promise a high degree of efficiency and scalability.

Figure ** shows the three types of conflict that can currently be handled: positive vs. negative authorization (i.e., being simultaneously permitted and forbidden from performing some action), positive vs. negative obligation (i.e., being both required and not required to perform some action), and positive obligation vs. negative authorization (i.e., being required to perform a forbidden action). We have developed policy deconfliction and harmonization algorithms within KAoS to allow policy conflicts to be detected and resolved even when the actors, actions, or targets of the policies are specified at vastly different levels of abstraction. These algorithms rely in part on a version of Stanford's Java Theorem Prover (http://www.ksl.stanford.edu/software/JTP/) that we have integrated with KAoS.

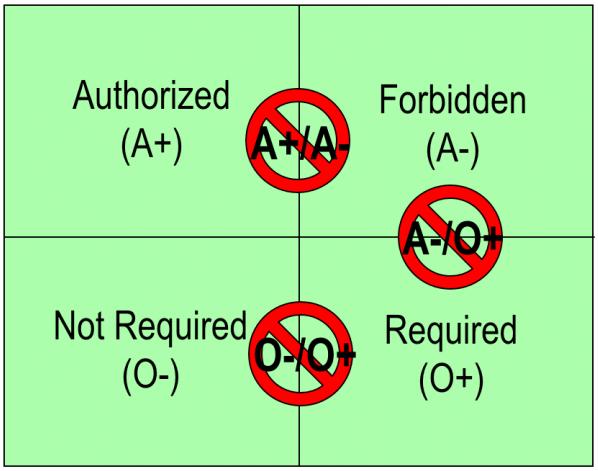

Figure 4: Three types of policy conflict.

Steps in policy conflict resolution. KAoS performs several steps in order to resolve policy conflicts:

1. DAML policy ontologies are loaded into JTP along with the set of DAML policies to be deconflicted.
2. A list of all policies in constructed and sorted according to user-defined criteria for policy precedence.
3. For each policy in the sorted list, iterate through all the elements with a lower precedence and check to see if there is a policy conflict. A policy conflict occurs if the two policies are instances of conflicting types and if a subsumption algorithm determines that the action classes that the two policies control are not disjoint.
4. The lower precedence policy from the conflicting pair of policies is removed from the list and the policy harmonization algorithm is invoked. It attempts to modify the policy with the lower precedence to the minimum degree necessary to resolve the conflict (if the policies are of equal precedence, a user may be

required to specify which policy will take precedence). The harmonization algorithm may generate zero, one or several new policies to replace the removed policy.

The newly constructed harmonized policies inherit the precedence and the time of last update from the removed policy, and a pointer to the original policy is maintained so that it can be recovered if necessary as policies continue to be added or deleted in the future.

5.4 Policy Enforcement

In order for the filter to be enforced, the newly-generated DAML class is provided, through the KAoS policy framework, to a policy enforcer. In applications to date, we have relied on several different kinds of enforcement mechanisms. Enforcement mechanisms built into the execution environment (e.g., OS or Virtual Machine level protection) are the most powerful sort, as they can generally be used to assure policy compliance for any agent or program running in that environment, regardless of how that agent or program was written. A second kind of enforcement mechanism takes the form of extensions to particular agent platform capabilities. Agents that participate in that platform are generally given more permissions to the degree they are able to make small adaptations in their agents to comply with policy requirements. Finally, a third type of enforcement mechanism is necessary for obligation policies. Because obligations cannot be enforced through preventive mechanisms, enforcers can only monitor agent behavior and determine after-the-fact whether a policy has been followed.

In CoAX, a message content *policy enforcer* used a message content *policy guard*, to test whether the policy applies to each message. This guard was developed using the Java Theorem Prover (JTP) developed at Stanford KSL [11]. The enforcer provides to the guard the class describing the filter for which the policy is defined along with the URIs of the DAML ontologies referenced by this filtering class. Subsequently, whenever messages being transmitted between the classes of agents covered by the policy are detected, the content of those messages (also represented in DAML) is given to guard for comparison to the message filter class. This test succeeds if the message content is inferred to be an instance of the filter class. If it is, and the policy is a positive authorization policy, then the message passes the filter, and the enforcer permits it to be sent to its destination. If the policy is a negative authorization policy and the test succeeds, then the message is blocked. The reasoning provided by JTP in conjunction with a set of axioms defining the semantics of the DAML language and the sets of ontologies referenced by the message filter class and the message being tested enables the necessary reasoning about toClass and hasValue restrictions of the policy.

In order for our semantic content filters to be tested against agent messages, the message content must be a DAML instance. For the CoAX demonstration, all agent messages were specified directly in DAML, by using DAML-S in conjunction with a *grounding* mechanism that wrapped the DAML message content (a string in RDF syntax) in a CoABS Grid message. If, however, the content of the message was in some other form, a mapping would need to be defined between the raw content of the message and a semantic encoding as a DAML description in order to use this approach to message filtering.

6. DYNAMIC TRANSFORMATION OF DATA FLOW BETWEEN AGENTS

Control of information release between agents may involve more than just the acceptance or rejection of messages based on policies and constraints. It is important, in some cases, to provide a mechanism to transparently modify the messages so they can satisfy the different policies for delivery in a secure and efficient way. Such a capability is especially important in critical scenarios such as military coalition operations or specialized sensor networks.

The FlexFeed framework relies on the automatic deployment of mobile agents with specialized data transformation capabilities to filter data streams so that they can satisfy policy constraints. These transformation agents act as policy enforcers that relay the data stream to the client agents. The framework is responsible for automatically customizing and deploying the agents that will constitute the distribution data path (or paths) between one or more source and sink agents.

The use of mobile agents as processing elements is an important feature of the FlexFeed framework, providing several additional capabilities such as: a) enabling the easy deployment of customized transformation code into intermediary nodes, and b) supporting the movement of transformation agents between nodes while retaining its state and redirecting resources (such as TCP connections) as needed.

Within the CoAX scenario, the need for data transformation arises when Arabello joins the coalition. As the scenario progresses, Arabello needs to obtain data from a Magnetic Anomaly Detection (MAD) sensor onboard an Australian ship. By default, policies do not allow any communication between agents in Arabello and agents in a particular country as shown in Figure 5.

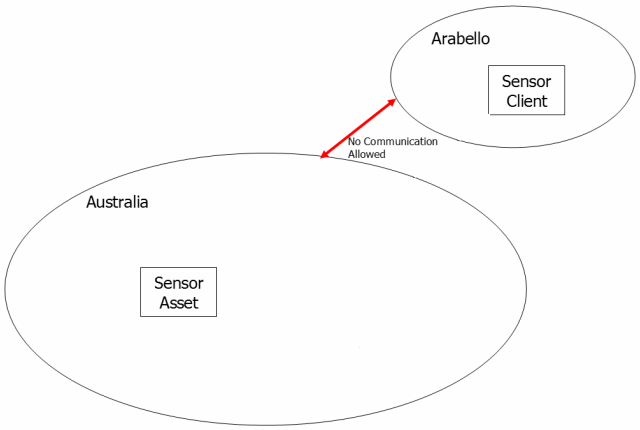

Figure 5: Australia and Arabello Domains Under Communication Restriction Policies

Direct communication between the to domains is not allowed but given the circumstances, the domain managers decide to allow a specific data feed to be provided as long as the full capabilities of the Australian sensor are not revealed to Arabello.

At this point, a new policy must be added to allow restricted communication between the nodes. KPAT is used by the Australian system administrator to create a new customized policy for data transfer. In this example, the policy refers to restrictions on video resolution given that the MAD data is transmitted as a series of images from the Australian sensor. To make this data available to Arabello, the resolution and the frame rate must be dropped, since the actual capabilities of the Australian sensor are classified.

The new policy is specialized to the type of data be transmitted. Because it is specific in terms of image resolution and frame rate, the policy can be added using a custom editor made available through a special pop-up menu in KPAT.

Figure 6 shows a screenshot of KPAT (in the back), with a superimposed screenshot of the custom policy editor for this example. Adding a specialized policy to this framework consists of having a DAML representation of the policy and a corresponding mobile agent for enforcement.

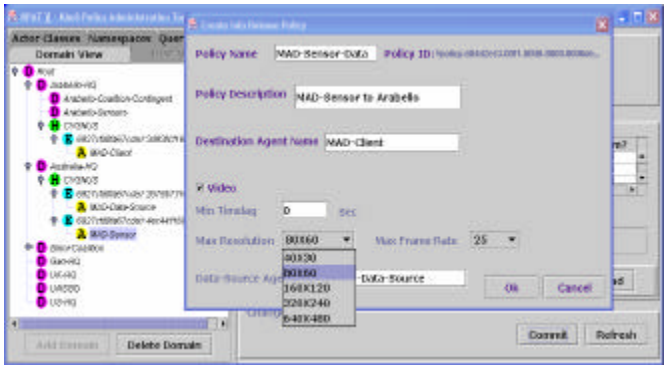

Figure 6: KPAT With a Custom Policy Editor For Video-style Data

A partial representation of the policy in DAML is shown in Figure 8 below.

The creation of an information release policy in this case will result on the immediate configuration and dispatch of a transformation agent to relay the data to the Arabello client.

Figure 7 shows a schematic representation of the deployment of the transformation agent. The transformation agent is configured to receive the video feed from the Australian sensor, reduce its resolution and relay the data to the Arabello client. The Arabello client is allowed to communicate only with the transformation agent.

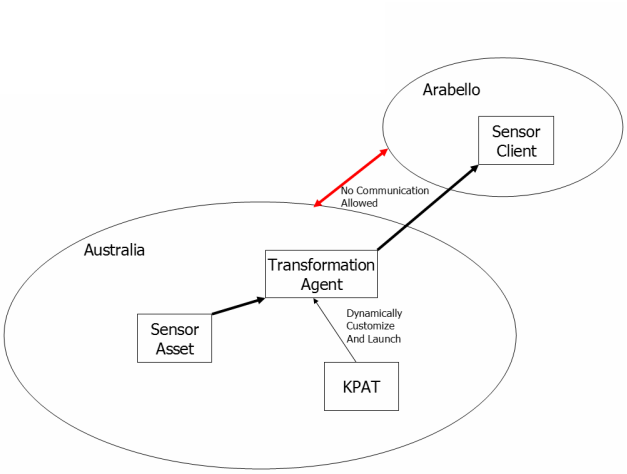

Figure 7: Automatic Deployment of the Transformation Agent as a Consequence of the Creation of the New Policy.

The transformation agent can be deployed to the sensor itself or to any intermediary node (another ship on the Australian domain, maybe) to distribute load within the domain. In the CoAX demo, the transformation agent is deployed directly to the Australian ship but the FlexFeed framework can, if needed, transparently determine the location to deploy the processing (or transformation) agent, for efficiency and load distribution.

One important aspect to be noted is that for the Arabello client,

```
...
  <rdf:Description rdf:about='#policy1-TargetList1'>
    <rdf:type rdf:resource='http://www.daml.org/2001/03/daml+oil#List'/>
    <NS0:first rdf:resource='http://ontology.coginst.uwf.edu/CoAX/Names/MeaureInstances.daml#Timelag0'/>
    <NS0:rest rdf:resource='http://www.daml.org/2001/03/daml+oil#nil'/>
  </rdf:Description>
  <rdf:Description rdf:about='#policy1-TargetList2'>
    <rdf:type rdf:resource='http://www.daml.org/2001/03/daml+oil#List'/>
    <NS0:first rdf:resource='http://ontology.coginst.uwf.edu/CoAX/Names/MeaureInstances.daml#Resolution80X60'/>
    <NS0:rest rdf:resource='http://www.daml.org/2001/03/daml+oil#nil'/>
  </rdf:Description>
...
  <rdf:Description rdf:about='#policy1-TargetList3'>
    <rdf:type rdf:resource='http://www.daml.org/2001/03/daml+oil#List'/>
    <NS0:rest rdf:resource='http://www.daml.org/2001/03/daml+oil#nil'/>
    <NS0:first rdf:resource='http://ontology.coginst.uwf.edu/CoAX/Names/MeaureInstances.daml#Rate25'/>
  </rdf:Description>
...
  <rdf:Description rdf:about='#policy1-Action'>
    <rdf:type rdf:resource='http://www.daml.org/2001/03/daml+oil#Class'/>
    <rdfs:subClassOf rdf:resource='http://ontology.coginst.uwf.edu/CoAX/CoAXAction.daml#VideoTransmittingAction'/>
...
  </rdf:Description>
  <rdf:Description rdf:about='#policy1-'>
    <NS1:hasDescription>MAD-Data to Arabello</NS1:hasDescription>
    <NS1:hasUpdateTimeStamp>1036700500015</NS1:hasUpdateTimeStamp>
    <NS1:controls rdf:resource='#policy1-Action'/>
    <NS1:hasName>MAD-Sensor-Data</NS1:hasName>
    <NS1:hasPriority>5</NS1:hasPriority>
    <rdf:type rdf:resource='http://ontology.coginst.uwf.edu/Policy.daml#PosAuthorizationPolicy'/>
  </rdf:Description>
</rdf:RDF>
```

Figure 8: DAML Representation of the information release policy

the whole process can be transparent. In the CoAX experiment, upon requesting a feed from the Australian ship, the domain administrator (a human) decides to create the new policy, which results in the deployment of the agents. It then notifies the Arabello client that the data is available from agent T (the transformation agent).

The FlexFeed framework can also hide this process from the client. In such an example, if a subsequent call is made from the Arabello client to the Australian sensor, since the information release policy is already in place, the transformation agent would be immediately deployed and data would start flowing to Arabello, with no need for human intervention. When the client requests the termination of the feed, the link is terminated and the transformation agent is simply discarded.

Figure 9 shows a screenshot of the MAD data in both ends of the link. In the back image the window shows how the data is available for agents within the Australian domain (higher resolution) and the front image represents the data received by the Arabello client (lower resolution).

The policy enforcement agent discussed in this example essentially provides three different types of data transformation: a) it can change image resolution, b) change frame rate and c) introduce a time lag, to prevent transmission of a real time video.

There are though, many other types of transformations that could be applied to the data. A transformation agent could, for instance, be implemented to hide sensitive targets or classified infrastructures from the image. This would be used to prevent the release of non necessary details to the requesting agent by blurring or editing the image appropriately. Another example is an agent that reduces the precision of coordinate values that are part of messages being transmitted.

7. SUMMARY AND FUTURE WORK

Both the semantic filtering and the dynamic transformation components were successfully demonstrated as part of the DARPA Coalition Agents Experiment (CoAX). Semantic filtering was used to protect sensitive data from Arabello from being released to the rest of the coalition members. Dynamic transformation was used to reduce the resolution of the data feed from a MAD sensor in the Australian domain to Arabello.

Policy-based control over data release and data transformation is extremely important in situations where multi-agent systems encompass more than one entity, administrative domain, organization, or country. Incorporation of mechanisms such as those described in this paper are essential to providing (human) system administrators the control and the confidence to use agent-based architectures in real-world scenarios

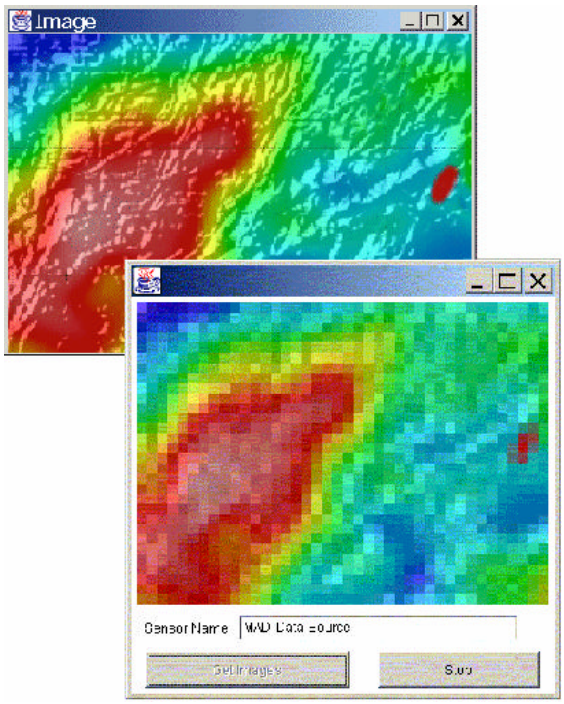

Figure 9: Data transformation for policy enforcement

.

We are investigating extending our semantic content filters in order to handle more complex restrictions than was possible using constraints on individual message properties. In many cases this pushes the limits of DAML expressiveness as well as requiring extensions to the set of axioms used by JTP in order to test for other types of restrictions. For example, in the CoAX domain, it would be useful to allow only contact reports where the object position is inside a certain region, specified by a set of latitude/longitude points. The mathematical relationship between a point and a polygonal region cannot be expressed directly in DAML. Furthermore, the computation required is best done by functional attachment to an axiom, rather than using logical inference. We are currently working on ways to bring this kind of filtering within the range of the overall policy enforcement mechanism.

We are also working on additional types of transformation agents and the corresponding policy editors for KPAT.

8. REFERENCES

[1] Allsopp, D., Beautement, P., Bradshaw, J. M., Durfee, E., Kirton, M., Knoblock, C., Suri, N., Tate, A., & Thompson, C. (2002). Coalition Agents eXperiement (CoAX): Multi-agent cooperation in an international coalition setting. A. Tate, J. Bradshaw, and M. Pechoucek (Eds.), Special issue of IEEE Intelligent Systems, 17(3), 26-35.

[2] Ankolenkar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D.L., McDermott, D., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T.R., and Sycara, K. (2002). DAML-S: Web Service Description for the

Semantic Web. Presented at The First International Semantic Web Conference (ISWC).

[3] Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., and Zeng, H. (2001, July). DAML-S: Semantic Markup for Web Services. Presented at International Semantic Web Working Symposium (SWWS).

[4] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. Scientific American.

[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] Bradshaw, J. M., Greaves, M., Holmback, H., Jansen, W., Karygiannis, T., Silverman, B., Suri, N., & Wong, A. (1999). Agents for the masses: Is it possible to make development of sophisticated agents simple enough to be practical? IEEE Intelligent Systems(March-April), 53-63.

[7] Bradshaw, J. M., Suri, N., Breedy, M. R., Canas, A., Davis, R., Ford, K. M., Hoffman, R., Jeffers, R., Kulkarni, S., Lott, J., Reichherzer, T., & Uszok, A. (2002). Terraforming cyberspace. In D. C. Marinescu & C. Lee (Ed.), Process Coordination and Ubiquitous Computing. (pp. 165-185). Boca Raton, FL: CRC Press.

[8] Carvalho, M. and Breedy, M. (2002) Supporting Flexible Data Feeds in Dynamic Sensor Grids Through Mobile Agents. Proceedings of the 6th International Conference on Mobile Agents (MA 2002). Berlin: Springer-Verlag.

[9] Global Info-Tek , Inc. DARPA CoABS Grid. On-line Reference: http://coabs.globalinfotek.com.

[10] Hendler, J. and McGuinness, D. L. (2000, November). The DARPA Agent Markup Language. IEEE Intelligent Systems. [Online]. 15 (6) , pp. 67-73.

[11] http://www.ksl.stanford.edu/software/JTP

[12] Kahn, M., & Cicalese, C. (2001). CoABS Grid Scalability Experiments. O. F. Rana (Ed.), Second International Workshop on Infrastructure for Scalable Multi-Agent Systems at the Fifth International Conference on Autonomous Agents. Montreal, CA, New York: ACM Press.

[13] Rathmell, R.A. (1999) A Coalition Force Scenario 'Binni - Gateway to the Golden Bowl of Africa', in Proceedings of the International Workshop on Knowledge-Based Planning for Coalition Forces, (ed. Tate, A.) pp. 115-125, Edinburgh, Scotland, 10th-11th May 1999.

[14] Sun Microsystems, Inc. Jini Network Technology. On-line Reference: http://wwws.sun.com/software/jini/.

[15] Suri, N., Bradshaw, J. M., Breedy, M. R., Groth, P. T., Hill, G. A., & Jeffers, R. (2000). Strong Mobility and Fine-Grained Resource Control in Nomads. Proceedings of the 2nd International Symposium on Agents Systems and Applications and the 4th International Symposium on Mobile Agents (ASA/MA 2000). Zurich, Switzerland, Berlin: Springer-Verlag,

[16] 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.

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