On Security in Open Multi-Agent Systems*

Lalana Kagal
University of Maryland
Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
lkagal1@cs.umbc.edu

Tim Finin
University of Maryland
Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
finin@cs.umbc.edu

Anupam Joshi
University of Maryland
Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
joshi@cs.umbc.edu

ABSTRACT

In open multi-agent systems agents must interact with other agents with which they are not familiar. In particular, an agent will receive requests and assertions from other agents and must decide how to act on the requests and assess the credibility of the assertions. In a closed environment, agents have well known and familiar transaction partners whose rights and credibility are known. The problem thus reduces to authentication which is the reliable identification of agents' true identity. In an open environment, however, agents must transact business even when knowing the true identities is uninformative. Decisions about who to believe and who to serve must be based on an agent's properties. These properties are established by proving them from an agent's credentials, beliefs of other agents and the appropriate security policies. In this paper we present an approach to some security problems in open multi-agent systems based on distributed trust, the delegation of permissions and distributed belief. Distributed trust management involves verifying if the requesting agent meets the required credentials for the request. As distributed trust management is inherently policy-based, our approach also includes a flexible policy language. Agents collect information that they require to make security decisions through distributed belief, which allows rules to be specified for accepting beliefs of other agents. We begin by describing our approach and the concepts on which it is built. Then we present a design that provides security functionality in a typical agent framework (FIPA) and describe initial work in its realization.

Keywords

security, trust, distributed trust management, policy based, multi-agent systems

1. INTRODUCTION

Though there has been some research in trust based security for multi-agent systems, generally multi-agent systems have always relied on traditional security schemes like access control lists, role based access control and public key infrastructure. These physical methods use system-based controls to verify the identity of an agent or process, explicitly enabling or restricting the ability to use, change, or view a computer resource. However these methods generally require some sort of central repository or control to provide authentication, need to store access control information for individual agents or groups of agents and need to know a priori whom to provide access control for. We believe that these schemes will not scale adequately or provide the increased flexibility required for emerging dynamic multi-agent systems that consist of an extremely large number of agents that are spread over a large geographic area like the agentcities project[1]. Hence we argue that it no longer makes sense to divide authorization into authentication and access control [17, 14].

We propose a security framework for multi-agent systems which is based on distributed trust management. Distributed trust management involves proving that an agent has the ability to access some service/resource solely by verifying that its credentials comply with the security policy of the requested service [17, 3]. These credentials include properties of the agent, for example, membership in certain organizations, age or host of the agent, recommendations and delegations by other agents.

The policy includes a set of rules that associate a required set of credentials with a certain ability or right; implying that only agents with the specified credentials can possess the ability. Our policy language provides constructs based on deontic concepts. These constructs are extremely flexible and allow different kinds of policies to be stated. This simple policy language is not tied to any specific application and allows domain dependent information to be added and reasoned over without any modification. Policies can be described in first order logic [19] and/or RDF-S [4]. The policy language includes a RDF-S ontology for specifying entities, objects, policies, credentials, and domain specific information. The policy engine, when given a policy, a knowledge base, and a request with an associated a set of credentials, is able to determine if the request is valid and meets the requirements of the security policy.

Though security policies associate credentials with actions or services, the process of verifying credentials is handled outside the system. This allows existing mechanisms like PKI, Kerberos etc. to be used. Another mechanism to check the properties of an agent is distributed belief. The policy language allows an agent to state rules for accepting believes of other agents. For example, ccn.com can be trusted with news related information. This provides a way for an agent to query other agents it trusts in order to verify the properties of the requesting agent.

2. DISTRIBUTED TRUST MANAGEMENT

[1]http://www.agentcities.org/

Our approach follows the trust management approach proposed by Blaze et al. [2, 3, 17], which formulates authorization as verifying whether an entity has the credentials that comply with the security policy governing the requested resource. These credentials include all properties of the agent including delegation assertions regarding the agent. Distributed trust is used as a way of authorizing an agent's requests based on the associated security policy and the current beliefs of the requested entity. Aspects of trust management include creating security policies, associating credentials with certain abilities and reasoning over these policies and credentials to decide the rights of an agent.

Trust is explained in terms of a relationship between a trustor, who trusts a certain entity, and a trustee, the trusted entity [8]. Based on his trust in the trustee, a trustor can decide whether the trustee should be allowed to access his resources and what rights should be granted. Therefore, trust plays an important role in deciding the access rights of a trustee. Using trust involves trust establishment and trust management. The basis for trust differs from entity to entity, leading to trust being distributed, as every entity has its own parameters for establishing and managing trust. Trust establishment is the process of deciding what the trust relationship with another entity should be. Trust management involves using the trust information, including recommendations from other trustees, to reason about authorization requests. In dynamic systems, every entity enforces its individual policies so a trust management component cannot use global policies or trust relationships to evaluate requests for authorization. This causes trust pertaining to each entity to be handled specifically in terms of its own policies, which requires principles of distributed trust.

In this work we view trust as being binary, i.e. an entity either *trusts* or *does not trust* another entity, which meets its security policy, to perform a certain action/service.

Our framework allows agents to delegate access rights, with restrictions attached, to other agents. An agent can delegate rights to any service it offers and delegate rights to any service it has been given the right to delegate the right to by an authorized agent. A delegation usually has constraints attached, such as one that limits the access to a certain period, or to whom the right can be re-delegated. A delegation consists of various information; delegator, right, constraints on delegatee, constraints on execution, constraints on re-delegation and time period. By using constraints on delegatee, the delegator can specify whom to delegate to. For example, a delegation could be conferred on all agents with certificates from a certain CA and registered with a certain platform. By restricting which of the delegatee can actually use the right, the delegator can prevent wrongful execution of the right. Constraints on re-delegation allow the delegator to decide whether the right can be re-delegated and to whom it can be re-delegated. We have developed rules that capture this information and enforce security by checking these constraints at the right time. We have separated the constraints on execution from the constraints on delegatee, to make delegation more flexible and its management more complete.

3. POLICY LANGUAGE

Our policy language is modeled on deontic logic and includes notions of rights, prohibitions, obligations and dispensations [11]. We believe that most policies can be expressed as what an entity can/cannot do and what it should/should not do in terms of actions, services, conversations etc., making our language capable of describing a large variety of policies ranging from security policies to conversation and behavior policies. The language is described in first order logic that allows for easy translation from/to RDF, DAML+OIL and OWL. We have also developed ontologies that enables the associated policy engine to interpret RDF-S policies.

The policy language has some domain independent ontologies but will also require specific domain ontologies. The former includes concepts for permissions, obligations, actions, speech acts, operators etc. The latter is a set of ontologies, shared by the agents in the system, which defines domain classes (person, file, deleteAFile, readBook) etc. and properties associated with the classes (age, num-pages, email).

Though the execution of actions is outside the policy engine, the policy language includes a representation of actions that allows more contextual information to be captured and allows for greater understanding of the action and its parameters. Similarly it allows domain dependent information to be added as conditions and reasons over them while making security decisions. It also allows complex conditions to be built from simple conditions using *and, or* and *not* and complex actions to be composed using action operators of *sequence, non-deterministic choice, once* and *iteration*. Using these actions and conditions, a policy maker is able to create policy objects. There are four kinds of policy objects: rights, obligations, prohibitions, and dispensations. Each policy object is a tuple of action and associated requirements. In order to associate these policy objects with users, the policy maker uses the *has* predicate creating rights, obligations etc. for the users.

Example 1 : A deployer of a platform specifies that agents from University of Maryland Baltimore County can register with the AMS.

has(X, right(register-with-ams, from(X, umbc)))

where, register-with-ams is an allowable action, umbc is a constant for University of Maryland Baltimore County and from(X, umbc) is a domain dependent condition specified by the deployer.

The language models four speech acts that can be used within the system to modify policies dynamically: delegate, revoke, cancel and request. In order to make correct policy decisions, it expects all relevant speech acts about the resources and users in its policies to be input to it.

The language also contains meta-policy specifications for conflict resolution. Meta-policies are policies about how policies are interpreted and how conflicts are resolved statically. Conflicts only occur if the policies are about the same action, on the same target but the modalities (right/prohibition, obligation/dispensation) are different. Meta policies in our system regulate conflicting policies statically in two ways; by specifying priorities and precedence relations [15]. Using a special construct, *overrides*, the priorities between any two rules can be set. It is possible to specify which modality holds precedence over the other in the meta-policies. The deployer of the platform/agent can associate a certain precedence for all policies associated with a set of actions or a set of agents satisfying a certain set of conditions.

Associated with the policy language is a policy engine that interprets and reasons over the policies and speech acts to make decisions about users rights and obligations.

4. DISTRIBUTED BELIEF

As described earlier distributed trust management involves proving that the requester of the action/service meets the security requirements of the requested entity. However proving that the credentials of the requester are valid is outside the scope of our system. We assume that third party tools are used to perform this verification whose result is then input to the system. We also provide another way for the agents to learn about the properties of other

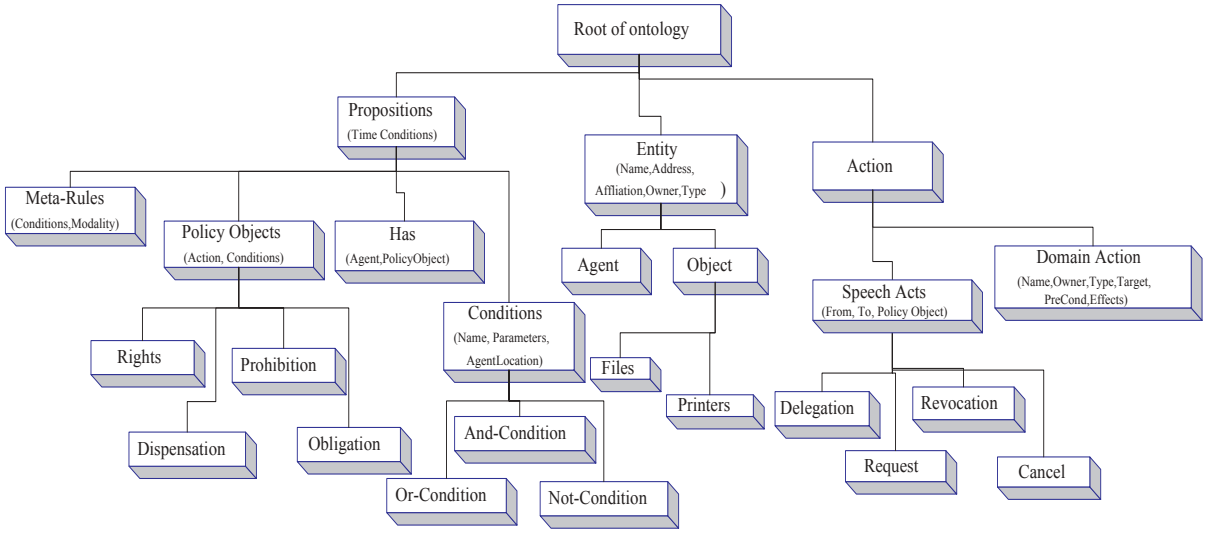

Figure 1: Ontology for Policies

agents – distributed belief. Agents can collect information that they require to make security decisions through distributed belief, which allows rules to be specified for accepting beliefs of other agents or other information sources in general. Using the policy language, the deployer of agents can describe rules for distributed belief in the security policy. The policy language includes two clauses for defining rules; *believe-true* for all information that is believed to be true based on certain conditions and similarly *believe-false* for information that is believed to be false based on some conditions.

Example 2 : The deployer of an agent decides that he believes what the ITTalks agent has to say about talks held at UMBC.

believe-true(property(X,Y), Conditions)

Conditions = and(says(ittalksAgent, property(X,Y)), and(talk(X), held-at(X, umbc)))

where, believe-true is a clause provided by the policy language and Conditions and property(X,Y) are any domain dependent conditions that can be defined by the deployer.

If the conditions associated with a 'value' in the believe-true clause are true then the system assumes that the 'value' is true. Similarly if the conditions associated with a 'value' in the believe-false clause are true then the system assumes that the 'value' is false. Conflicts are handled through meta-policies by setting priorities to rules or deciding which modality holds precedence. If a value is true/false then it can be used to prove/disprove the credentials of agents.

5. DESIGN OVERVIEW

Our model provides security functionality for FIPA-based agent platforms using distributed trust management. According to principles of trust management [17], a request is authorized if the requesters credentials meet the security requirements associated with the requested object. However the mechanism does not specify how these requirements are described or by whom, how or where the requirements are stored and who carries out the processing. In our research we address these issues in detail and describe an infrastructure for authorization in open multi-agent systems.

Our approach is similar to role based access control in that a user's access rights are computed from its properties. However, we use additional ontologies that include not just role hierarchies but any properties and constraints expressed in a semantic language including elements of both description logics and declarative rules. For example, there could a rule specifying that if an agent in a meeting room is using the projector, it is probably a presenter and should be allowed to use the computer too. In this way, rights can be assigned dynamically without creating a new role. Similarly, rights can be revoked from a user without changing his/her role, making this approach more flexible and maintainable than role based access control.

This framework, based on FIPA [6] specifications, assumes that agents exist within platforms. An agent platform consists of a white page service called Agent Management Services (AMS), a yellow page service known as Directory Facilitator (DF) and a communication service known as Agent Communication Channel (ACC). We extend the functionality of the AMS and the DF to manage authorization for the platform in order to restrict the usage of the platforms functions. Similarly, an authorization mechanism is also built into agents. Communication between the different entities is encrypted using public key cryptography. Though digital certificates are used the system does not expect the entities to be authenticated and uses certificates for encrypting messages only.

5.1 Security Classification

We provide three security categories for agents and services; *private*, *secure* or *open*. An agent, while registering with a platform, can send some security information to the AMS specifying its security category as either private, secure or open. A *private* agent's Agent Identifier (AID) is not displayed to any other agent by the AMS, a *secure* agent has to send some access control information so that the AMS can filter requests to the agent and and an *open* agent is visible to all agents. In the same way, while registering its services with a DF, an agent can choose a category for each service. For example, an agent A can register as an open agent with the AMS and register two services with the DF, a GPS service which is open and a navigator service which is secure. Agent A also specifies that only agents with certificates signed by Verisign can access the navigator service.

5.2 Policy Specification

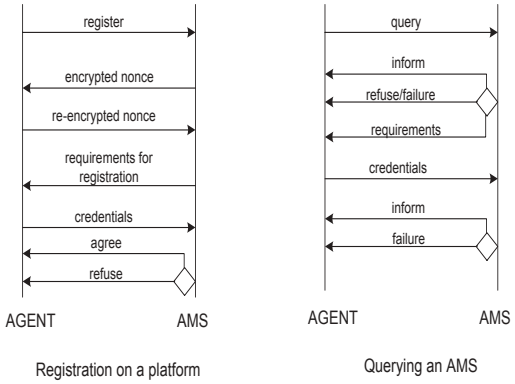

Figure 2: Interaction Protocols for the Agent Management System

The deployer of the platform decides which group of agents should be able to register with and query the AMS and the DF. There are four main functions performed on a platform that this system takes into consideration; registering with the AMS, querying the AMS, registering a service with the DF, and querying the DF for a service. The deployer uses the policy language to specify the properties required by an agent to perform each of these functions. These properties could be certificates, login-passphrase, membership in a group or even an emails from a trusted entity.

In the same manner, the deployer of an agent specifies its security policy by identifies the services of the agent and associating properties required for their use. The deployer of the agent can also specify communication acts that an agent responds to and the security requirements that the sender must meet.

Though the policy language allows different kinds of credentials to be specified like certificates, roles, groups etc., the actual verification of these credentials is outside the system. This allows different kinds of authentication and verification schemes to be hooked into this mechanism for greater flexibility. The system also allows distributed belief to be used to verify credentials.

5.3 Authorization Module for an AMS

The AMS is responsible for managing agents and maintaining a searchable table for agent identifiers (AID). Our system considers two functions of the AMS, registration of agents and querying for AIDs. Figure 2 illustrates the interaction protocols for each function.

When an agent wants to register with the AMS it encrypts its request including its category with its private key and sends it to the AMS along with its digital certificate. To verify the validity of the request, the AMS starts the *handshaking protocol* that is common in Public Key Systems. It sends the agent a small message, a nonce, encrypted with the agents public key, and attaches the platforms certificate, to the address specified by the requesting agent. This is not only done so that both parties can verify each other, but also in order to verify the agents location, prevent spoofing and securely exchange information. The agent decrypts the nonce using the platforms key and sends it back encrypted with its own key. On getting the valid reply from the agent, the AMS sends back requirements for registration encrypted with the agents public key[2]. It is assumed that the agent understands the ontology of the requirements or has a way of translating these requirements into an ontology it understands. The agent gathers the required credentials and sends them to the AMS encrypted with its private key. The AMS verifies the credentials using third party tools or through its distributed belief policies to decide if the agent can register.

If the agent does not have the right to register with the AMS, its request is denied. Otherwise the registration is allowed to go through. The AMS stores the certificate (public key) of the agent along with its agent identifier into the AID table.

A registered agent can query for other agents by sending a query message encrypted with its own private key to the AMS. The AMS checks the credentials required for querying are a subset of the credentials required for registration. If they are, the AMS satisfies the query and sends back the reply encrypted with the agents public key. Otherwise the AMS sends back the requirements for querying As the case of registration, the agent has to gather the credentials and send it to the AMS. After verifying the credentials, the AMS responds to the query with a list of AIDs encrypted with the agents public key. The AMS informs the agent about all the agents that are either in the open category or the secure category for which the agent fulfills the required conditions for access. If there are certain AIDs of secure agents that match the query but whose requirements the requesting agent does not meet the AMS could send back the combination of requirements the agent would need to view these agents. However this could be a potential violation of privacy and we are currently debating whether or not to add this functionality.

5.4 Authorization Module for DF

Associated with the DF is a security policy that specifies the requirements associated with registering and querying for services. Figure 3 illustrates the different interaction protocols related to the DF. An agent registered with the AMS can register a service with the DF by sending a register agent message with the description of the service and the category of the service (open, secure and private). If the service is secure the agent includes the policy associated with the service. The message is encrypted with the agents private key and contains the public key. The DF checks if the agent is registered by sending the public key to the AMS. If the agent is not previously registered with the AMS the register message is denied. If registered and if its security policy is a subset of that of the AMS the DF allows the registration to go through. If not a subset the DF sends the requirements that the agent has the show to register encrypted with the agents public key[2]. The agent sends the required credentials encrypted with its key. The DF verifies the credentials using third party tools or through distributed belief and then registers the service. As part of the registration process the DF stores a pointer to the location of the agents entry into the AID table of the AMS.

To query the DF for services, the agent sends a signed query message to the DF. The DF verifies the message and the checks the category of the services that fulfill the search query. The DF sends back all open services that meet the query and those secure services whose requirements the agent fulfills. The message is encrypted by the agents public key. If the agent does not meet the requirements for a secure service, the DF could send back the list of required credentials. However this could be a potential privacy concern and is currently being debated.

We introduce a new performative to enable agents to get *tickets* to access services that they have access to. An agent sends a *getTicket* message to the DF encrypted with its private key. If the agent has the right to access the service, the DF constructs a *ticket* that the agent can uses to access the service on the associated service agent. A ticket is a tuple made of service identifier and the

[2]It is also possible to assume that all platforms have a web presence and publish their policies on the web enabling this interaction to be skipped.

requesting agents public key encrypted with the platforms private key. The ticket is then encrypted with the agents public key and sent to the agent. While accessing the service on the service agent, the requesting agent attaches the ticket, so the service agent knows that the requester's credentials have been verified.

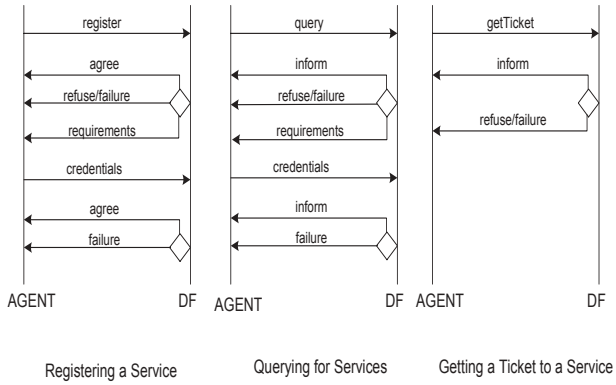

Figure 3: Interaction Protocols for the Directory Facilitator

5.5 Authorization Module for Agent

Every agent has two modes of operation as an owner of a service and as a requester of a service.

Security on the agents side can be handled in multiple ways. An agent can decide to register its services as *open* or *private* on the DF, so that the agent itself is completely responsible for access control. The second way is for the agent to categorize its services as *secure* and specify the access control conditions in the DF. If the agent trusts the DF completely, it can rely on the DF to handle access control and the agent need not have a security module at all. If the agent does not trust the DF, it can implement its own security module for stricter access control. In this case, after the requests are filtered by the DF, they can be re-verified by the service agent. The service agent expects all requests to be encrypted with the agents public key and to include a ticket from the DF. The service agent verifies the ticket from the DF and uses its security policy to decide if the agent meets its requirements for accessing that particular service. If all the requirements are met, the service agent sends the result back encrypted with the requesters public key. The agent does not go through the handshaking procedure because the sender has a valid ticket from the platform. However inspite of the ticket from the DF, a service agent may decide not to honor a certain request, because there may be certain additional constraints it requires that the requesting agent fails to meet. Please refer to Figure 4 for the interaction protocols of agents.

The deployer of an agent can also specify a security policy for each communication act, in fact creating conversation policies. Each time an agent receives a message it checks the associated security policy before responding to the message.

6. ONGOING RESEARCH

Our current approach is based on two security systems previously developed by us which are based on distributed trust management - an agent-based supply chain management application [12] and an agent-mediated pervasive computing environment [13, 20]. During their implementation we have refined our trust management concepts and developed several programs in logic for handling the propagation of delegation, and validating requests.

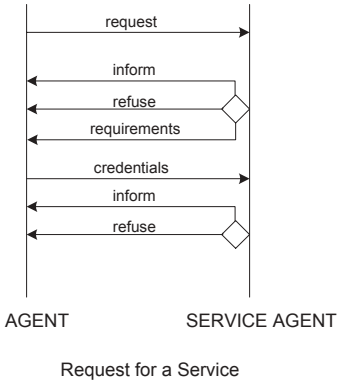

Figure 4: Interaction Protocols for Agents

Our policy language [11] currently supports deontic concepts of rights, obligations, prohibitions, and dispensations. It also accepts fours speech acts that manipulate the access rights of agents; delegate, revoke, request and cancel. Meta-policies are used for conflict resolution. They permit positive or negative modality to be given precedence for a set of policies and priorities to be set between conflicting rules. The system also includes comprehensive delegation management and allows flexible policies to be represented in both first order logic, namely Prolog, and RDF-S. Our system when given a policy, a set of credentials of an entity and the requested action/service is able to correctly deduce if the action is permitted or not. The policy language is currently being refined to include mechanisms for distributed belief. We are also working on adding security functionality to the Agent Management System, Directory Facilitator and Agent classes of JADE [1], a FIPA [6] based agent platform.

7. RELATED WORK

There has been a lot of interesting work in security for multi-agent systems, and in this section we describe some research projects that are most relevant to ours.

Wong and Sycara describe the design of a security infrastructure for multi-agent systems [21]. Their work is based on RETSINA, a resuable multi-agent infrastructure. The authors describe several threats associated with multi-agent systems with respect to the RETSINA framework; corrupted agent naming servers or matchmakers, insecure communication channels, and insecure delegations. To prevent the threat of corrupted ANSs or matchmakers, the authors believe it is necessary to use only trusted ANSs and matchmakers that behave as they should, by only servicing valid requests, inserting/removing entries from their database in a way that is consistent with the request and giving responses that are consistent with their databases. As a way of counteracting lack of accountability, all agents should be given proofs of identity that cannot be forged and deployers of agents should be made responsible for the actions of their agents. Communication channels should be made secure and agents should be made to prove that they are delegatees of whom they claim to be. Certificates are used to link agents to actions and deployers to agents for accountability. The authors describe mechanisms for agent key certification and revocation, in which the deployer interacts with the ACA. Then they discuss protocols for registration, unregistration and lookup. To handle insecure communication channels, the authors plan to add SSL (secure socket layer) underneath their agent communication layer.

In their paper 'Distributed Trust in Open Multi-Agent Systems', the authors build on earlier work by Herzberg et al [9] to define an infrastructure for distributed trust in multi-agent systems [16]. Following Herzberg's assumptions, the authors think that identity is not required for trust management, and that there is no need for a centralized certificate mechanism or trusted third parties. This work is based on the use of certificates. Most role based access control mechanisms map users' identities to role. However this is not the approach used in this work; an agent uses its policy to map another agent to a role, based on the latter's certificates [9]. Any agent can be a certificate issuer, and may not be globally trusted. An issuer is trusted when it can provide sufficient certificates from other issuers to satisfy the requester's policy. An agent could have several certificates certifying its capabilities and its performance. These certificates will be from other agents that have used the agent's services. However these certifying agents may not be globally trusted. If an agent X needs to find a particular service, it sends a request to the MatchMaker in a system like RETSINA [21]. The MatchMaker will return a list of matching agents and their certificates. The requesting agent will reason about these certificates to decide which agents can be trusted. The policy will define rules for deciding trust levels based on the certificates. To solve the problem of authorizing accessing agents, every agent has as part of its architecture an access control mechanism. This component helps the agent decide which services should be accessible to a certain agent. The access control component uses certificates to map an accessing agent to a role, and then uses role based access control to decide its access rights.

In his paper, Hu explains how to build up an agent oriented PKI and demonstrates some delegation mechanisms for it [10]. In this agent oriented PKI, there are two types of certificates; identity certificates for humans and their agent, and authorization certificates for humans and agents. Authorization certificates are used to represent authorizations by entities. These include the public key for the granting entity, the public key of the entity receiving the authorization, the actual authorization (access right), re-delegation bit, and the validation period. However, the re-delegation bit always set to 1, because the author does not have any fail-proof method of preventing re-delegation. Though there is a difference between trust between humans and agents and between agents, the author models them in the same way. Hu also describes 3 types of delegations; chain-ruled, threshold, and conditional. In chain-ruled the access rights are delegated in a cascading manner. Threshold delegation allows an entity to delegate to multiple subjects. These subjects must co-operate with each other to perform the delegation. When the subject has to satisfy certain conditions in oder to use the delegation, it is called conditional delegation. As authorizations can be re-delegated, they form delegation networks. The verification process checks that every entity in the delegation network has the authority to re-delegate, that all the authorizations are within the validity period, and that none of the required certificates have been revoked. However, this study does not include mechanisms for handling revocation of certificates. The verification can either be done by a Trusted Third Party or the original issuer agent. Usually the service guardian authorizes other agents to use the service, who in turn authorize other agents. Generally the original issuer agent is the verifying authority as well. Rules for verifying an authority are specified as part of the delegation policies within the original issuer agent's rule base. If a Trusted Third Party is responsible for verification of authority validity, then it is also responsible for all the service access control. The author has included several performatives for human/agent identity certificate management and human/agent authorization certificate management. Hu also describes how these performatives are encoded in XML for agent communication.

Poslad et al. describe the security and trust notions currently part of the FIPA specifications and point out some of its strengths and weaknesses [18]. The FIPA security specifications were started in 1998, but are still not complete and have actually been made obsolete by FIPA. The authors believe that security is domain dependent and that it is not possible to have a general security architecture which is suitable for all applications. The authors describe the trust models existing in FIPA. All agents that want to use services or provide services in a platform must register with the platform's Agent Management System (AMS). The AMS is trusted and maintains the identity of all registered agents. However as authentication is not mandated, spoofing is possible. AMS is responsible for the life cycle for all agents in the platform and agents must report all significant changes to the AMS and allow the AMS to control their life cycles. However an agent need not obey orders from the AMS, causing the AMS to take some other course of action like using an external API or de-registering the agent. Agents also register their capabilities/services with the Directory Facilitator (DF). There are no specifications about this registration, so a malicious agent could cause a lot of damage by registering non-existent services, registering wrong service descriptions etc. FIPA does not define how accessing agents can specify their preferences. There exists a trust relationship between the Agent Communication Channel (ACC) and registered agents. The ACC is trusted to transmit the messages in a timely fashion and to maintain the integrity of the messages. The FIPA security model [7] defines mechanisms for keeping messages private, mechanisms to check the integrity of messages and authentication messages. This model extends the functionality of AMS and DF and introduces an entity called Agent Platform Security Manager (APSM), which is responsible for maintaining security in the platform. The AMS uses public key infrastructure mechanisms for authenticating agents wishing to register with it. This raises issues related to PKI [5]. The agents define additional security parameters as part of their service descriptions which they register with the DF. The current specifications also include some suggestions for secure Agent Communication Language (ACL) communications, mainly the envelop construct. Certain keywords like authentication, non-repudiation etc can be used to express a level of security. When an agent requests a service, it is the responsibility of the message transport layer to encapsulate the messages based on these levels. The semantics of these keywords are provided by the platform. The authors propose certain requirements for adding security to FIPA systems, including authentication of agents by middle agents (AMS and DF) when writing to directories accessed via middle agents, use of private channel to send messages, and authentication of middle agents by agents for bi-directional trust.

8. CONTRIBUTIONS

Our system addresses the challenges associated with MAS, namely, corrupted AMS and DF, insecure communication, insecure delegations, lack of accountability, access control for foreign agents, and lack of central control. The model manages corrupted naming and matchmaking services by using a PKI handshaking protocol between the agent and the AMS to verify validity of both parties. All messages are encrypted according to Public Key Infrastructure. However we do not use these certificates for authenticating agents but for exchanging messages securely. Our infrastructure allows foreign or unknown agents access into the system using trust management. When an unknown agent tries to register with a platform, the platform checks the agents credentials, and decides its rights with that platform based on the security policy. Multiagent systems are inherently decentralized and it is not possible to have a

central database of access rights or policies. This is not a problem in our system as no central information is required. The policy is enforced at two levels; at the platform level, where access to the AMS and DF is controlled and at the agent level, where an agent can specify who can access its services.

The policy language is composed of domain dependent information and domain independent information. It provides specifications for representing domain independent information (constraints, actions etc.) allowing the deployers to use specific information that it has no prior knowledge of, but can still reason over while making decisions. *Types* of policy objects to be specified. For example, all the rights on a certain resource, prohibition from printing to any color printers on the fifth floor, and the right to delete all the files belonging to your colleague. Though the policy specifications allow these kinds of policy objects, based on properties of actions, the policy engine does not support this functionality as yet. The policy language supports individual policies as well as group and role based policies to be specified in a uniform manner. The policy engine includes strong delegation management making it useful for dynamic systems, consisting of transient resources and users, and distributed systems, in which creating comprehensive policies may be time consuming.

9. SUMMARY

In this paper we present the design for a security framework for multi-agent systems based on trust management. We believe that other interesting concepts like reputations and obligations can also be built in once the basic framework is developed. This approach is particularly useful in open environment in which agents must interact with other agents with which they are not familiar. We believe our model addresses several prominent security issues associated with these agent environments and provides a comprehensive trust based solution.

10. REFERENCES

[1] F. Bellifemine, A. Poggi, and G. Rimassa. JADE - A FIPA-compliant Agent Framework. In *Proceedings of PAAM'99, London, April 1999, pagg.97-108*, 1999.

[2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System Version. Internet RFC 2704, September 1999., 1999.

[3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The Role of Trust Management in Distributed Systems. *Secure Internet Programming, LNCS vol. 1603, Springer, Berlin, 1999, pages 185-210*, 1999.

[4] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft 30 April 2002, http://www.w3.org/TR/rdf-schema/, 2002.

[5] C. Ellison and B. Schneier. Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. *Computer Security Journal*, 16, 2000.

[6] F. for Intelligent Physical Agents. FIPA Specification. http://www.fipa.org/spec/, 2001.

[7] F. for Physical Intelligent Agents. FIPA 98 Specifications Part 10, Version 1.0, Agent Security Management, 1998.

[8] T. Grandison and M. Sloman. A Survey Of Trust In Internet Applications. *IEEE Communications Surveys, Fourth Quarter 2000*, 2000.

[9] A. Herzberg, Y. Mass, J.Mihaeli, D.Naor, and Y. Ravid. Access Control meets Public Key Infrastructure : Or Assigning Roles to Strangers. In *Proceedings of 2000 IEEE Symposium on Security and Privacy, Oakland, May 2000*, 2000.

[10] Y.-J. Hu. Some thoughts on Agent Trust and Delegation. In *Proceedings of Autonomous Agents 2001*, 2001.

[11] L. Kagal. Rei : A Policy Language for the Me-Centric Project. HP Labs Technical Report, 2002.

[12] L. Kagal, T. Finin, and Y. Peng. A Framework for Distributed Trust Management. In *Proceedings of IJCAI-01 Workshop on Autonomy, Delegation and Control*, 2001.

[13] L. Kagal, J. Undercoffer, F. Perich, A. Joshi, and T. Finin. A Security Architecture Based on Trust Management for Pervasive Computing Systems. In *Proceedings of Grace Hopper Celebration of Women in Computing 2002*, 2001.

[14] N. Li, B. N. Grosof, and J. Feigenbaum. A Practically Implementable and Tractable Delegation Logic. In *Proceedings of IEEE Symp. on Security and Privacy, held Oakland, CA, USA, May 2000*, 2000.

[15] E. C. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, November/December 1999.

[16] Y. Mass and O. Shehory. Distributed Trust in Open Multi Agent Systems. In *Workshop on Deception, Fraud and Trust in Agent Societies, Autonomous Agents 2000*, 2000.

[17] M.Blaze, J.Feigenbaum, and J.Lacy. Decentralized Trust Management. In *Proceedings of IEEE Conference on Privacy and Security*, 1996.

[18] S. Poslad and M. Calisti. Towards Improved Trust and Security in FIPA Agent Platforms. In *Autonomous Agents 2000 Workshop on Deception, Fraud and Trust in Agent Societies, Spain, 2000*, 2000.

[19] S. I. o. C. S. Swedish Institute. SICStus Prolog. http://www.sics.se/ sicstus/, 2001.

[20] J. Undercoffer, A. Cedilnik, F. Perich, L. Kagal, and A. Joshi. A Secure Infrastructure for Service Discovery and Management in Pervasive Computing. *ACM MONET : The Journal of Special Issues on Mobility of Systems, Users, Data and Computing*, 2002.

[21] H. Wong and K. Sycara. Adding Security and Trust to Multi-Agent Systems. In *Proceedings of Autonomous Agents '99 Workshop on Deception, Fraud, and Trust in Agent Societies, May, 1999, pp. 149 - 161*, 1999.