Developing Secure Agent Systems Using Delegation Based Trust Management *

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

We present an approach to some security problems in multi-agent systems based on distributed trust and the delegation of permissions, and credibility. We assume an open environment in which 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 – the reliable identification of agents' true identity. In an open environment, however, agents must transact business even when knowing the true identities is un-informative. 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, delegation assertions, and the appropriate security policy. We begin by describing our approach and the concepts on which it is built. Then we present a design that provides security functions (authorization and credibility assessment) in a typical agent framework (FIPA) and describe initial work in its realization using the semantic web language DAML+OIL.

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 and need to store access control information for individual agents or groups of agents. We believe that these schemes will not scale adequately or provide the increased flexibility required for emerging dynamic multi-agent systems that consists of an extremely large number of agents that are spread over a large geographic area [11] like the agentcities project[1]. Hence we argue that it no longer makes sense to divide authorization into authentication and access control [16, 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 [16, 2]. 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 process of verifying the credentials is itself under the security policy of the verifying agent. 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. Our trust management system includes a trust ontology for specifying entities or principals, policies, credentials, a mechanism for verifying credentials and a mechanism for checking if the credentials conform to the policy. 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.

Agents communicate their beliefs with each other for trust management. Beliefs are exchanged in terms of delegations, credentials, abilities of other agents and trust values. An agent will reason about beliefs (its own and of other agents) and policies while making authorization decisions.

This framework, based on FIPA specifications [6], addresses many of the security threats generally associated with Multi-Agent Systems (MAS) [20]. The challenges usually associated with MAS are corrupted naming (Agent Management System) and matchmaking (Directory Facilitator) services, insecure communication, insecure delegations, lack of accountability, access control for foreign agents, and lack of central control [20].

2. 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 [20]. 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, insecure delegations, and lack of accountability. 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

*This work was supported by NSF Awards IIS 9875433 and CCR 0070802, and the Defense Advanced Research Projects Agency under contract F30602-00-2-0 591 AO K528.

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

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 [8] to define an infrastructure for distributed trust in multi-agent systems [15]. 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 [8]. 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 [20]. 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 [9]. 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 [17]. 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 [3]. 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.

3. DESIGN

This model provides security based on distributed trust management for open, dynamic agent platforms, with methods for intra-platform and inter-platform security.

Agents are authorized to access a certain service if they have the required credentials. Our work 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.

We extend the functionality of the Agent Management System (AMS) and the Directory Facilitator (DF) to manage security for the platform, as not all agents should be able to register on a particular platform or use a certain DF. Similarly, agents are also given some access control ability. The AMS, DF and agents follow certain security policies to decide the access rights of requesting agents.

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 delegation mechanism is able to thwart any invalid or insecure delegations. Only agents with the right to delegate can actually make valid delegations that change the access rights of other agents. All agents are held accountable for their actions because they have to sign all service queries and requests with their own private key. As there is a unique private key public key pair, once an agent signs a request, the agent can be held accountable. 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 individually at the entity processing the request. 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.

Agents are able to delegate their rights in a controlled and secure fashion. For example, if agent A delegates some service to agent B, and agent B tries to delegate this service to agent C, then the second delegation will fail as agent A did not give agent B the ability to redelegate.

The agents use a semantic language like DAML+OIL [4] as an ontology language. DAML+OIL is an ontology language for marking up resources, and is basically being developed for the realization of the Semantic Web[2]. The agents express security information including credentials, delegations, and policies in DAML+OIL making it easier for other agents to interpret them correctly.

3.1 Security Classification

We classify security into two levels depending on where it is enforced: platform or agent. In platform security, the AMS and DF have additional security features. The AMS can decide whether or not to allow an agent to register, search or use its other functions. Similarly, the DF can also decide whether to allow an agent to register, modify or search for agents based on certain access control information. An agent, while registering with a platform, can send some security information to the AMS specifying its security category; *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. Similarly, while registering its services with a DF, the 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 agent B, with certain credentials, can access the navigator service. In agent security, the agent uses a policy to decide how to further validate service requests.

3.2 Platform Security

In platform security the AMS and DF use distributed trust management principles to authorize requests to their services.

3.2.1 Security Module for an AMS

When an agent wants to register with the AMS, it signs it's request and sends it to the AMS, along with its digital identity certificate. The AMS verifies the certificate based on the rules. The rules could be of the form, an entity X of the organization Y with a certificate from trusted Certificate Authority CA(Y) is valid. Or there could be a rule saying, for all certificates from organization Z, calculate certification path and verify with the CA. If the certificate is valid, the AMS checks the signature. The AMS uses its policies to decide what access rights the agent has on the platform.

If the agent does not have the right to register with the AMS, its request is denied. If the agent does have the right to register with the AMS, 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 can now go ahead and verify the platforms certificate. It then replies to the AMS with the same nonce encrypted with the platforms public key. On receiving this, the AMS creates a *trust certificate* containing the platform related rights of the agent, the associated public key, time validity and other relevant information and sends it back. This certificate is valid only for a short time, after which the agent has to start the registration process again. This period is directly based on the *level of trust* associated with the agent or in fact the agent's *reputation* in the platform. Using a trust certificate enables the AMS and DF to skip the rechecking of the agents' credentials everytime the agent tries to use the services of the platform.

After creating the trust certificate, the AMS will inform 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. During the period of validity of the trust certificate, the agent can make requests to access the AMSs services. These requests have to be signed. The AMS does not need to check all the credentials of the agent, but only verifies that the agent has the right to the requested service.

3.2.2 Security Module for a DF

[2]W3C's Ontology Wrapper Language (OWL) is based on DAML+OIL

After obtaining a trust certificate from the AMS of a platform, the agent can access various services of the AMS and the DFs. Using the trust certificate, an agent can register its services with the DF, if the platforms policy allows that particular agent to use the DF. This service registration message is signed with the agents private key, and acts as a digital signature. This forces agents to be accountable for their actions. The DF verifies the trust certificate and checks that the trust certificate is valid and belongs to the agent. It retrieves the agents public key from the trust certificate, and checks the signature of the registration message. If the certificate states that the agent has the right to register, the DF proceeds with the registration. An agent can register its different services under different categories. This service description is also in DAML+OIL, making the searching more semantic and more flexible. To query the DF, the agent sends a signed query message to the DF. The DF verifies the message and the checks the category of the service that fulfills the search query, the conditions attached if a secure service, and the access rights of the requester before sending back any results. These results are encrypted with the agents public key, which is associated with the trust certificate.

Agents can 'delegate' authorization ability to the DF if they share domain ontology. If an agent cannot use the DF for making authorization decisions on its behalf, then the agent has to contain a trust management engine and interpret its own policies. This makes the presence of the engine in an agent optional, allowing agents to run on smaller, lightweight, devices. The AMS/DF has a list of conditions that an agent must satisfy in order to contact a particular agent or use a particular service. However the AMS and DF need to understand the service agent's[3] policy or have access to its knowledge base. It is upto the service agent to make sure that these conditions are accurate and conform to its policy. In some cases, the AMS or DF cannot understand the associated conditions. Then, based on the policy of the platform, the AMS and DF can decide to reject all requests for the agent or service or accept all requests and forward them to the appropriate service agent for interpretation.

3.3 Agent Security

The authorization decisions carried out by individual agents for access to their services comprises agent security.

3.3.1 Security Module for Agent

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

Owner 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.

Requester of a service

After an agent receives a matching list of services as a result of its DF query, it tries to execute one of them. The agent sends a request to the service agent and attaches its identity certificate and trust certificate. This message is encrypted using the agents private key. The receiving agent carries out similar reasoning as the AMS, by going through its certificate verification rules to verify the identity certificate and trust certificate. If both the certificates are valid, it verifies the signature. It then uses its security policy to decide if the agent meets its requirements for accessing that particular service. If all the checks are valid, then the receiving agent sends the result back encrypted with the senders public key. The agent does not go through the handshaking procedure because the sender has a valid trust certificate from the platform. Even after the platform checks by the AMS and 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.

[3]The agent controlling a service is known as service agent

4. INTER PLATFORM SECURITY

If an agent is already registered with a platform and wants to access the AMS or DF on another platform, the agent should send along with its identity certificate, its current trust certificate, which contains information about its access rights. The remote platform decides the agents rights in the normal fashion based on its own security policy, and may take into consideration the platform that the agent is currently registered on.

DF's of different platforms can be accessed if they register with each other through principles of *federations of DF* [6]. If an agent is searching for a particular service, and its DF cannot find any matching service, the DF will forward the request with the trust certificate to the other DFs registered with it. These DFs will process the agents request as normal and return the results.

5. VERIFICATION OF CREDENTIALS

Credentials are properties of agents that are described in a semantic language and signed by other agents. Delegations are special credentials and are discussed in detail in Section 6. In order to accept credentials of other agents, an agent must be able to verify these credentials. Verification can be carried out in the following ways

- Simple Verification : In this scheme, a service agent expects all the credentials necessarily at the time of request. In order to use its services, a requesting agent must send all required credentials along with the request for service. The service agent will check its knowledge base, and question other agents about their beliefs in order to verify the credentials. Suppose agent A has an alarm service which requires that requesters be AAAI members. The security policy of agent A also states that the agent XYZ should be trusted to verify AAAI certificates. An agent B sends A a request to use the service along with its certificate from the AAAI CA. This certificate states that the bearer of this certificate is a member of AAAI. Agent A asks agent XYZ to verify the certificate. If the certificate is valid then agent B is authorized to use the alarm service. If agent B did not send the required certificate or sent an invalid certificate, its request would be denied.

- Negotiation : Certain service agents may provide a more interactive requesting mechanism. If the requested agent does not provide the correct credentials to access the service, the service agent asks the requester for specific additional credentials. For example, a service agent A only allows employees of XYZ Pvt. Ltd. to access its services, and accepts delegations from these employees. Agent B approaches agent A with a credential from AAAI. Agent A decides that the credential is not good enough and asks the agent B to prove that it is an employee of XYZ or if B has a delegation from an employee. Agent B possesses a delegation from Bob who is an employee of XYZ and sends this delegation to A. A ver-

ifies the delegation and the chain of delegations and decides to authorize agent B's request.

- Third Party : Some service agents do not have the resources to verify credentials and so request trusted third parties to handle the verification on their behalf. Suppose a trusted agent, C, did have the resources and the inclination to help agent A, agent A would send B's credentials to C to be verified and would trust C's response. C could either use simple verification or negotiation to verify these credentials.

6. DELEGATIONS

An agent has the ability to make any delegation, but whether it is honored depends on various factors, including the security policy, the agent's rights, and the rights of the agents ahead it in the delegation chain. Agents are not prevented from making delegations, but the delegations by unauthorized agents are considered invalid. Only agents with the ability to delegate can make valid delegations. Valid delegations change access rights of other agents. The right to delegate is defined implicitly and explicitly. Implicitly, an agent can delegate rights to any service it offers. Explicitly, an agent that has been given the right to delegate by an authorized agent can perform valid delegations, as long as the delegation fulfills the constraints of the previous delegation. This forms a chain of constraints; the agent at the end of the chain must satisfy all the constraints associated with the delegations in the chain. Our delegation mechanism, written in logic, verifies that the requesting agent satisfies all the constraints of the delegations before it in the chain.

Our framework allows certain authorized agents to delegate access rights, with restrictions attached, to other agents. 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.

6.1 Delegation Management

Though delegation is very important for the propagation of trust, managing delegations in a distributed and dynamic environment is rather difficult. Consider the following example, an agent (delegator), who is delegated a certain right, delegates it to another agent (delegatee) and goes down immediately. The delegatee asks to use the certain resource and presents its delegation certificate. However this request cannot be validated because the delegator's ability to delegate cannot be checked.

We suggest three schemes for managing delegations

Delegation Chain The previous example can be solved by making the delegator attach its own delegation certificate to the newly created delegation before sending it to the delegatee. This means that every agent will have to store a chain of delegation certificates leading to its own delegation, in order to validate its delegation. This is not feasible because each chain could be very long and there could be several delegations for every agent. To reduce the number of certificates in a chain, certificate reduction could be used [1], but the original delegator may not be accessible.

Centralized Delegation To avoid handling and processing chains of delegations, all delegations can be addressed to the service agent or the agent platform responsible for the service agent. However this scheme has two problems; it is rather centralized and the delegator may not be able to access either the service agent or the agent platform at the time of the delegation.

Delegations on the Web The last scheme is to continue using delegation chains, but instead of storing the chains within the agent, the chains could be stored on web pages. In order to prove it has a certain ability, an agent could point to a certain delegation on its delegation page. This delegation in turn would refer to a delegation on the page of the agent who made the delegator. By traversing this delegations, the agent platform and/or service agent would be able to verify the delegation and decide whether or not to authorize the request.

7. TRUST PERFORMATIVES AND INTERACTION PROTOCOLS

FIPA is based on speech acts, predicate logic and public ontologies. Speech acts are ways of communicating or expressing oneself [18]. A speech act only succeeds if it is understood by the recipient as intended. However FIPA does not include the speech acts required for trust management. As part of this security initiative, several speech acts, that are common to distributed trust domains, will be modeled. In this framework, agents will use certain speech acts to explain their intent; delegating, requesting etc. For example, "I delegate to you the ability to access my files for one hour", or "I request you to delegate to me the use of your workstation". These statements contain a lot of information that needs to be captured. An ontology, grounded in DAML+OIL will be used to describe these speech acts. This ontology will enable the audience to correctly interpret the speech act and understand its purpose. FIPA Communicative Acts describe a set of "utterances" used in multi-agent systems, and FIPA Interaction Protocol specifies the order of messages exchanged. Though most of the communication between the agents can be modeled with existing FIPA performative, we believe certain additional performatives are required for agent security and trust.

The performatives that will be added are Request Permission, Delegate, Request Verification, and Credential Required.

- Request Permission
  The action of asking another agent for permission to access a certain service.
- Delegate
  The action of delegating to another agent or group of agents the ability to perform a certain action on a certain service.
- Credential Required
  The action of asking the recipient to provide additional credentials. The content is the credential required and this performative is the response to a request where the recipient did not provide the correct credentials.
- Request Verification
  The action of asking the recipient to verify credentials supplied by an agent requesting access to the sender's service.

Using existing FIPA communicative acts and the performatives described above, we describe the interaction protocols for trust management in our system.

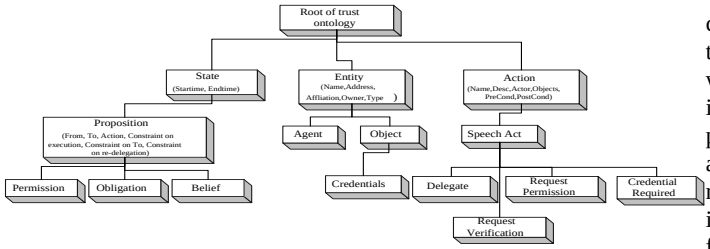

Figure 1: Our trust ontology as a class hierarchy

- Request Interaction Protocol
  This interaction protocol allows the initiator to request the use of another agent's service. The initiator sends the recipient a request message. Similar to FIPA, some responses are not-understood, refuse, agree, failure, inform-done, and inform-ref [5]. However depending on the kind of verification being performed by the recipient, the responses could also include Credential Required. The sender would now have to resend its request with the new credentials in order to gain access to the service.

- Request Permission Interaction Protocol
  An agent uses this protocol to request another agent to delegate certain abilities to it. The initiator starts the protocol by sending the recipient a Request Permission message. The responses from the recipient include not-understood, refuse, agree, failure, and Delegate.

- Request Verification Interaction Protocol
  The initiator uses this protocol when it is unable to verify some credentials and requires the recipient to verify the credentials on its behalf. The initiator starts this interaction protocol by sending a Request Verification message. The valid responses to this message are not-understood, refuse, agree, failure, inform-done, and inform-ref.

8. ONTOLOGIES

Our infrastructure uses ontologies expressed in DAML+OIL to represent security information and policies in a multi-agent system.

We have designed an ontology for trust and security information in this system, which is illustrated in Figure One. The root of the ontology is divided into State, Entity and Action. State contains all information pertaining to the current state. It currently has one subclass, Proposition, which is further sub-classified into Permission, Obligation, and Belief. Propositions are clauses that have a truth value in the system. An Entity could either be an Agent or an Object. An object can be extended to define domain specific resources like credentials, files, computers, printers, etc. An Action is associated with a set of Objects or resources. Speech acts like Requests and Delegations are extensions of Actions.

The ontology specific to agent systems extends the main trust ontology with information related to FIPA platforms; register an agent, deregister an agent, search, create, agent service, etc. as actions and certificates, platform address, network address, network protocol used etc. as objects. Figure Two shows part of the Agent ontology.

9. POLICY

The security policies are based on the Agent ontology. Each platform and agent follows a security policy. A security policy may contain rules for verifying certificates and credentials, access control, and delegation. Rules for verifying certificates could specify which certificate authorities are trusted, and the procedure involved in verifying different kinds of certificates, based on the CA, principal, agent etc. Rules for access control will state the credentials an agent must have for a certain access right. The policy also contains rules that describe the way delegations and revocations propagate in the system, how re-delegations are handled, how prohibitions affect access control and delegations and how revocations should be managed. For example, if a delegation is revoked, should all the agents that the delegatee delegated to, lose the access right as well or can they keep it and whether a prohibition is given priority over a delegation while deciding access rights.

Our *default policy* defines certain rules about the propagation of delegations so that all constraints in the delegation chain are applied before an agent can gain access to a service. If a certain link in the delegation chain fails or the right is revoked, the rest of the chain after this failed link loses the access right as well. This default policy also includes rules for the mechanisms of credential verification and belief management.

10. PREVIOUS WORK

We have previously developed two security systems based on distributed trust management - an agent-based supply chain management application [12] and an agent-mediated pervasive computing environment [13, 19]. 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.

10.1 Security for Supply Chain Management Systems

We successfully implemented a trust based framework for the Extended Enterprise COalition for Integrated Collaborative Manufacturing Systems (EECOMS) project, which is aimed at providing a set of technologies for integrated supply chain and business to business electronic commerce [10]. A supply chain management system consists of groups of buyers and sellers that need to open up their internal systems to each other in a secure way. In other words, a supply chain management system consists of a network of heterogeneous agents that interact to perform certain actions that may or may not need authorization. The main problem is guaranteeing the authenticity of requests between these agents, whether within a company or between one or more companies.

Our system sets up authorization and delegation rules, so that the information in the SCM may be accessed only by authorized agents. Special intelligent agents called *security agents* are required for authentication and authorization within a particular domain, and are trusted within the company and by the companys buyers and sellers. They also represent the company in some sense. The security agents of a company enforce the company policy. This policy describes certain rules for rights, delegation and for reasoning about them. The policy is not changed frequently and usually involves human intervention. Agents within a company possess an identity certificate that is signed by a trusted Certificate Authority. Agents within a company can be authenticated by the security agents through their ID certificates.

In order to allow the buyer's employees to access certain information within its company, the security agent of the seller gives the security agent of the buyer the permission to access that information, and the ability to delegate this right. To propagate this trust within its own company, the seller's security agent delegates this right to some of its employees based on the policy. Depending on

```
<?xml version="1.0" encoding="UTF-8" ?>

<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:dtrust = "http://daml.umbc.edu/ontologies/trust-ont#"
>
<daml:Ontology>
      <daml:Class rdf:ID="Date">
          <daml:equivalentTo>http://daml.umbc.edu/ontologies/
calendar#Date</daml:equivalentTo>
      </daml:Class>
      <daml:Class rdf:ID="String">
          <daml:equivalentTo>http://www.daml.org/2001/03/
daml+oil#Literal</daml:equivalentTo>
      </daml:Class>
</daml:Ontology>


<!-- SubClass of Objects; Certificate  -->
<rdfs:Class rdf:ID="Certificate">
      <rdfs:subClassOf rdf:resource="dtrust:Object"/>
          <rdfs:label>Certificate</rdfs:label>
          <rdfs:comment>
            This subclass contains information about Certificates
          </rdfs:comment>
          <daml:Restriction>
            <daml:onProperty rdf:resource="dtrust:Affiliation"/>
            <daml:toClass rdf:resource="#Organizations"/>
          </daml:Restriction>
</rdfs:Class>

<!-- Properties of Certificates -->
<rdf:Property rdf:ID="CA">
     <rdfs:domain rdf:resource="dtrust:Agent"/>
</rdf:Property>
<rdf:Property rdf:ID="Principal">
    <rdfs:domain rdf:resource="dtrust:Agent"/>
</rdf:Property>
<!-- more properties  ..... -->

<!-- SubClass of Certificates; ID, Trust, Delegation  -->
<rdfs:Class rdf:ID="IDCertificate">
      <rdfs:subClassOf rdf:resource="#Cerificate"/>
      <rdfs:label>IDCertificate</rdfs:label>
      <rdfs:comment>
          This subclass contains information about ID Certificates
      </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="TrustCertificate">
      <rdfs:subClassOf rdf:resource="#Cerificate"/>
      <rdfs:label>TrustCertificate</rdfs:label>
      <rdfs:comment>
        This subclass contains information about Trust
        Certificates
      </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="DelegationCertificate">
      <rdfs:subClassOf rdf:resource="#Cerificate"/>
      <rdfs:label>DelegationCertificate</rdfs:label>
      <rdfs:comment>
          This subclass contains information about Delegation
          Certificates
      </rdfs:comment>
</rdfs:Class>
```

```
<!-- Properties for Trust Certificate -->
<rdf:Property rdf:ID="Roles">
     <rdfs:domain rdf:resource="dtrust:Object"/>
</rdf:Property>
<rdf:Property rdf:ID="PublicKey">
     <rdfs:domain rdf:resource="dtrust:Object"/>
</rdf:Property>
<rdf:Property rdf:ID="StartDateTime">
     <rdfs:domain rdf:resource="#Date"/>
</rdf:Property>
<rdf:Property rdf:ID="EndDateTime">
     <rdfs:domain rdf:resource="#Date"/>
</rdf:Property>
<!-- more properties ...-->

<!-- Subclass of Object -->
<rdfs:Class rdf:ID="Organization">
     <rdfs:subClassOf rdf:resource="dtrust:Object"/>
     <rdfs:label>Organization</rdfs:label>
      <rdfs:comment>
        This subclass contains information about organizations
      </rdfs:comment>
</rdfs:Class>


<!-- Subclass of Actions; RegisterWithAMS -->
<rdfs:Class rdf:ID="RegisterWithAMS">
        <rdfs:subClassOf rdf:resource="dtrust:Action"/>
        <rdfs:label>RegisterWithAMS</rdfs:label>
     <daml:Restriction>
        <daml:onProperty rdf:resource="dtrust:Actor"/>
        <daml:toClass rdf:resource="dtrust:Agent"/>
     </daml:Restriction>
</rdfs:Class>

<!-- Properties of RegisterWithAMS -->
<rdf:Property rdf:ID="IDCertificate">
        <rdfs:range rdf:resource="#IDCertificate"/>
</rdf:Property>
<rdf:Property rdf:ID="RegisterMessage">
        <rdfs:range rdf:resource="#String"/>
</rdf:Property>
<!-- more properties ... -->

<!-- Subclass of Actions; QueryDF -->
<rdfs:Class rdf:ID="QueryDF">
        <rdfs:subClassOf rdf:resource="dtrust:Action"/>
        <rdfs:label>QueryDF</rdfs:label>
     <daml:Restriction>
        <daml:onProperty rdf:resource="dtrust:Actor"/>
        <daml:toClass rdf:resource="dtrust:Agent"/>
     </daml:Restriction>
</rdfs:Class>

<!-- Properties of QueryDF -->
<rdf:Property rdf:ID="TrustCertificate">
        <rdfs:range rdf:resource="#TrustCertificate"/>
</rdf:Property>
<rdf:Property rdf:ID="QueryString">
        <rdfs:range rdf:resource="#String"/>
</rdf:Property>
<!-- more properties ... -->


</rdf:RDF>
```

Figure 2: This image shows a portion of the Agent System Ontology. Registration of an agent, deregistration of an agent, querying a DF, etc. are all subclasses of the Action class in our Trust Ontology. Similarly, certificates, addresses, organizations etc. are subclasses of the Object class in our Trust Ontology.

the previous delegations, the employees can further delegate this right to other employees, forming a chain of delegation from the buyer's security agent to the seller's security agent to the seller's employees. If at any point a delegation fails or is revoked the access cannot go through. The same holds if the situation is reversed and the supplier gives the buyer access to some of its resources. Delegation chains should always trace back to a security agent to be valid. Security agents are responsible for all accesses originating from its company and act as gateways. All access to information outside the company must go through a security agent. This agent will authenticate the requester, check the delegation chain and verify that the requester has the right to access the requested information. The security agent creates an authorization certificate for the requesting agent, that the requesting agent can use for access.

This framework led us to view trust management as a very effective method for resolving several issues related to security in distributed systems.

10.2 Security for Pervasive Systems

We have designed and implemented Vigil, a security framework, which provides security and access control in pervasive systems [13]. Vigil has been optimized to work in *SmartSpaces*, which is a specific instance of pervasive environments. A SmartSpace environment provides services and resources, that users can access using some short range wireless communications such as Bluetooth, IEEE 802.11, or Infrared, via any hand-held device, within a Vigil can also be used in wired systems, but the focal point of our research is the security in dynamic, mobile systems. Vigil is designed so that clients can move, attach, detach, and re-attach at any point within the framework.

Our infrastructure is designed to minimize the load on portable devices and provide a media independent infrastructure and communication protocol for the provision of services. Vigil, in addition to solving the issue of controlling access to services in a *SmartSpace*, also accommodates users that are foreign entities, that is entities that are not known to the system in advance. In many conventional systems, access rights are static; agents are not able to request permission to access a Service to which they are not preauthorized. To overcome these issues, we have incorporated the *Vigil Security Agent*. This Security Agent allows agents to ask for access permission and other agents to actually delegate rights that they have. This extends the security policy in a secure manner, as only agents that have the permission to delegate, can actually delegate.

The Vigil system is divided into *SmartSpaces*, and each SmartSpace uses one or more security agents to maintain security. The Security Agent is responsible for maintaining distributed trust in the Vigil system. It enforces the security policy of the organization or SmartSpace. It interprets the policy to provide controlled access

to Services and uses distributed trust as a more flexible and easily extensible policy based mechanism. There is generally a global policy associated with the organization and a local policy associated with a SmartSpace. All security agents in the organization will enforce the global policy and will additionally enforce a local policy, which is specific to the Space. A policy includes rules for role assignment, rules for access control, and rules for delegation and revocation.

The Security Agent uses a knowledge base and sophisticated reasoning techniques to handle security and distributed trust. On initialization, it reads the policy and stores it in a Prolog knowledge base. All requests are translated into Prolog, and the knowledge base is queried. The policy contains *permissions* which are access rights associated with roles, and *prohibitions* which are interpreted as negative access rights. The policy also contains rules for role assignments, access control and delegation. A user has the ability to access a service if the user has not been prohibited from accessing the service by an authorized entity and if it either has the role based access right or if some authorized entity has delegated this right to it. An entity can only delegate an access right that it has the ability to delegate.

When a user needs to access a service that it does not have the right to access, it requests another user, who has the right, or the service itself, for the permission to access the Service. If the entity requested does have the permission to delegate the access to the Service, the entity sends a delegate message, signed by its own private key, along with its certificate, to the Security Agent and the requester. The Security Agent checks the roles of the delegator and the delegatee and ensures that the delegator has the right to delegate, and that the delegation follows the security policy. It then adds the permission for the Client to access the Service, but sets a very short period of validity for the permission. Once this period is over, The Security Agent has to reprocess the delegation. This is very useful incase of revoked certificates, delegations or rights. If any one entity in the delegation chain loses the permission, then it is propagated down the chain very quickly, till everyone after the entity loses the ability. Everytime a Service Broker asks about the delegated rights of the client, the Security Agent sends back only valid permissions.

11. SUMMARY

In this paper we present the design for a security framework for multi-agent systems based on trust management, the delegation of permissions and credibility. 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. Research in security for multi-agent systems often tends to focus on a limited subset of the security challenges of MAS. We believe our model addresses several prominent security issues associated with these agent environments and provides a comprehensive trust based solution.

12. REFERENCES

[1] Tuomas Aura. Distributed Access-Rights Managements with Delegations Certificates. *Secure Internet Programming*, pages 211–235, 1999.

[2] 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.

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

[4] Ian Horrocks et al. DAML+OIL Language Specifications. http://www.daml.org/ 2000/12/daml+oil-index, 2001.

[5] Foundation for Intelligent Physical Agents. Interaction Protocol Specifications. http://www.fipa.org/repository/ips.html.

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

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

[8] 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.

[9] Yuh-Jong Hu. Some thoughts on Agent Trust and Delegation. In *Proceedings of Autonomous Agents 2001*, 2001.

[10] Ingersoll Rand (Woodcliff Lake, NJ) and QAD (Carpenteria, CA) and Berclain Group (Schaumburg, IL) and IBM Corporation (Somers, NY). CIIMPLEX Consortium, Consortium for Integrated Intelligent Manufacturing PLanning and EXecution. http://www.ciimplex.org, 2000.

[11] Lalana Kagal, Tim Finin, and Anupam Joshi. Trust based security for pervasive computing enviroments. In *IEEE Communications, December 2001*, 2001.

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

[13] Lalana Kagal, Jeffrey Undercoffer, Filip Perich, Anupam Joshi, and Tim 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] Ninghui Li, Benjamin N. Grosof, and Joan 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] Yosi Mass and Onn Shehory. Distributed Trust in Open Multi Agent Systems. In *Workshop on Deception, Fraud and Trust in Agent Societies, Autonomous Agents 2000*, 2000.

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

[17] Stefan Poslad and Monique 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.

[18] J. R. Searle. Speech Acts : An essay in the Philosophy of Language. Cambridge University Press, 1969.

[19] Jefferey Undercoffer, Andrej Cedilnik, Filip Perich, Lalana Kagal, and Anupam 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.

[20] H.C. 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.