

STUDENT PAPER: A Delegation-based Distributed Trust Model for Multi Agent Systems 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
£nin@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, obligations 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 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, delegation assertions, and the appropriate security policy. We begin by describing our approach and the concepts on which it is built. We then describe its application in two implemented systems – an agent-based supply chain management application and an agent-mediated pervasive computing environment. Finally 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

*This research was supported by the DARPA DAML contracts F30602-00-2-0591 and NSF awards IIS 9875433 and CCR 0070802

Though there has been some research in trust based security for multi-agent systems, generally multi-agent systems have always relied on rather simple security methods such as access control lists and secure communication channels. We propose a model, that extends previous work in this area [26, 24], as a complete security infrastructure for such systems. This framework, based on FIPA specifications [9], addresses many of the security threats generally associated with MAS [26]. 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.

In this model, agents are uniquely identified by digital certificates, which includes their owner, organization, name and even *role*. Security is enforced at two levels: the platform level and the agent level. We extend the functionality of the Agent Management System (AMS) and the Directory Facilitator (DF) to manage security. Not all agents should be able to register on a particular platform or use a certain DF. Similarly, each agent is also given some access control ability. The AMS, DF and agents follow certain security policies to decide access rights of requesting agents.

DARPA Agent Markup Language (DAML) [6] is a way of marking up information to make it machine-readable and provide richer information to agents. We can express security information and policies in DAML+OIL making it easier for agents to interpret them correctly.

2. RELATED WORK

There has been a lot of interesting research in the area of security for distributed systems. PGP [28] provides a simple way of sending secure email using a *web of trust*, without exchanging a key and without a central authority. In PGP, a keyholder (an individual associated with a public/private key pair) learns about the public keys of others through introductions from trusted friends. The largest problem associated with PGP is key distribution and management. The Simple Public Key Infrastructure (SPKI) was the first proposed standard for distributed trust management [13]. This solution, though simple and elegant, includes only a rudi-

mentary notion of delegation, which is crucial to the development of *distributed trust*. Personal security agent [10] uses a public key infrastructure for agent systems based on trust hierarchies. This provides interoperability between different agent systems using PKI.

Role Based Access Control [20, 27] is one of the better known methods for access control. In this approach, entities are assigned *roles*, each of which has a set of associated rights. Unfortunately, this is difficult to apply for systems in which it is not possible to assign roles to all users in advance. Also it is typically not possible to change access rights associated with a particular entity without modifying the roles. Herzberg et. al. [11] describe an approach to access control for foreign entities by using property based credentials and establishment of trust based on policies.

The application of trust management to enforce security in distributed systems has been widely recognized [3, 5, 12, 26, 21, 16] as an interesting alternative. A logical approach for trust, authentication and access control was defined by Abadi et. al. [1]. PolicyMaker [22] was able to interpret policies and answer questions about access rights. Unfortunately, the development of policy is complicated and not easy for non programmers. This poses quite a problem, as it is generally non programmers who are responsible for defining and maintaining such policies. Keynote [2] was designed along the same lines as PolicyMaker and it requires the policies and queries to be written in a specific language.

Li et. al. [19, 18] define a language and protocol for delegation based access control, which tends to focus on authorization based on properties of entities. It specifies a language for authorization in open systems, which is used to represent policies, credentials and request. This complex logic language includes a delegation clause that has a delegation depth to control how many re-delegations are possible.

The above mentioned models are very powerful, however they do not meet all our requirements for security in multi-agent systems. Generally security systems should not only authenticate and authorize known users, but also provide access control to unknown users based on their credentials. They should allow users to delegate securely their rights and beliefs to other users and provide a flexible mechanism for this delegation including the notion of restricted re-delegation.

3. DISTRIBUTED TRUST MODEL

We view trust management as the *establishment of trust relationships* instead of its classical meaning of quantifying trust. This approach involves articulating policies for user authentication, access control, and delegation; assigning security credentials to individuals; allowing entities to modify access rights of other entities by delegating or deferring their access rights to third parties and revoking rights as well; and providing access control by checking if the initiators' credentials fulfill the policies [22, 23]. An agent is assigned some generic rights deduced by reasoning over its credentials, the security policy and delegations from other agents [11]. Agents can also make requests to access other services. Appropriate agents with these access rights can decide to delegate the requested right to them. An agent can dele-

gate all rights that it has the permission to delegate. Rights can also be revoked; so rights are no longer static, but can change based on delegations and revocations.

An agent can access any service that it has either an axiomatic right specified in the policy or that has been delegated to it. It can also delegate this right to other agents, if it has been given the right to subsequently delegate it. A delegation itself is a right which can be delegated. So, an agent could be given the right to perform some action but not to further delegate it or given the right to some action and the right to delegate it, or the right to delegate some action but not the right to execute it.

Rights or privileges can be given to trusted agents who can then be held responsible for the actions of agents to whom they may subsequently delegate the right to. So the agents will only delegate to agents that they trust. This forms a *delegation chain*. If any agent along this chain fails to meet the requirements associated with a delegated right, the chain is broken and all agents following this invalid link are not permitted to perform the action associated with the right.

We tried to solve security issues in systems that consist of widely distributed resources and agents [15]. We have implemented security infrastructures for distributed systems including supply chain management systems [16] and pervasive computing environments [17].

3.1 Security for Supply Chain Management

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 [14]. 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. Our system sets up authorization and delegation rules, so that this information 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 companies buyers and sellers. They also represent the company in some sense. The security agent of the buyer can give the security agent of the supplier the permission to access certain information, and the ability to delegate this right. The supplier's security agent can delegate this right to some of its employees based on the policy. This security agent is responsible for all accesses coming from its company. The employees can further delegate this right forming a chain of delegation from the buyer to the supplier to its employees. If at any point the 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. So, delegation chains should always trace back to a security agent to be valid.

The 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 group/company or between

one or more companies. The security agents of a company follow the company policy. This policy describes certain rules for rights, delegation and reasoning about them. These security agents enforce the security policy of the company. The policy is not changed frequently and usually involves human intervention.

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

3.2 Security for Pervasive Systems

We have designed and implemented Vigil, a security framework, which provides security and access control in pervasive systems [17]. 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 pre-authorized. 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 in case 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. Every time a Service Broker asks about the delegated rights of the client, the Security Agent sends back only valid permissions.

A user can also *revoke* rights that it has delegated by sending the appropriate message to the Security Agent. The Security Agent removes the permission for the delegated entity, and when a Service Broker asks about the delegated entity, it is informed of the revoked right. So revocations progress rapidly through the system.

We drew on our research work on the above mentioned projects along with our experience with agents and agent systems [7, 25, 8] to design a trust model for multi-agent systems based on delegations.

4. OBJECTIVES

We would like to provide a framework for security based on distributed trust for distributed open agent platforms, with a method for intra-platform and inter-platform security. Our infrastructure couples existing security methods like Simple Public Key Infrastructure (SPKI) and Role Based Access Control (RBAC) with notions of *distributed trust management* (refer to Section 3 for more information) and *domain role assignment*. We view trust management as the *establishment of trust relationships* instead of its classical meaning of quantifying trust. Domain role assignment is the assigning of a temporary domain based role to an agent based on its credentials. This is only done so that we can apply the principles of RBAC but keep in check the size of the access control lists. We do not use strict RBAC, but an extension by which we can change access rights associated with agents dynamically without changing their roles to existing roles or new roles.

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. Insecure communication is managed using SSL, which has already been implemented for the most widely used agent platforms, e.g., JADE [24]. 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. Using the domain role assignment technique, our infrastructure allows foreign or unknown agents access into the system. When an unknown agent tries to register in the platform, the AMS checks its credentials, based on which the agent is assigned a domain based role and the access rights associated with that role. An agent could be in more than one role at a time, and is given all the access rights associated with all its roles. For example, an agent could be both a graduate student and a research assistant. An agent has the right to a certain service, if the right is associated with one of its roles, if the right was delegated to it by an agent who had the ability to delegate, and if the agent was not prohibited by an authorized agent from using that service. Multiagent systems suffer from lack on central control; there is no central database of access rights or policies. But in our system, this is not a problem, because the policy is enforced at two levels; at the platform access the AMS and DF is controlled and at the agent level, 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 it fail, because agent A has not given agent B the ability to redelegate access. Revocation of rights is also allowed at all levels.

One of the most important issues related with security is the overhead in administration. We tried to mitigate this by making agents responsible for using the security features correctly please refer to section 6.3.

All state changing propositions are logged for future auditing purposes and evaluation of a possible security breach. This helps find faults in security mechanism and this information could be used to form *reputation* of agents that make invalid accesses, delegations, prohibitions or revocations.

5. 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 can send some access control 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, *secure* agent has to send some access control information so that the AMS can filter requests to the agent and *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 authenticate agents and how to authorize their service requests.

6. DESIGN OVERVIEW

We are integrating security and trust into the FIPA platform using a semantic language like DAML+OIL and extending well understood concepts of Simple Public Key Infrastructure (SPKI) [13] and Role Based Access Control (RBAC) [27].

The Agent Communication Language messages in this system use DAML+OIL as the content language. So agents communicate with each other and the agent platforms through DAML+OIL.

On starting up, a platform reads its security policy and the AMS and DF retrieve rules about verification of certificates, authentication, role assignment and access control. These rules are based on the agent identity, its organization, its CA, any reputation it may have, and its platform, if already registered with another platform.

All communication between the agent and the rest of the world is encrypted, the agent has to sign its messages and the receivers send back replies encrypted with the agents public key.

6.1 Security Module for an AMS

When an agent wants to register with the AMS, it signs its 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 role the agent should be assigned and after that what rights the agent possesses based on the role and the agent itself. This information is cached, for easy checking for future requests. This cache is cleared periodically.

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, 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

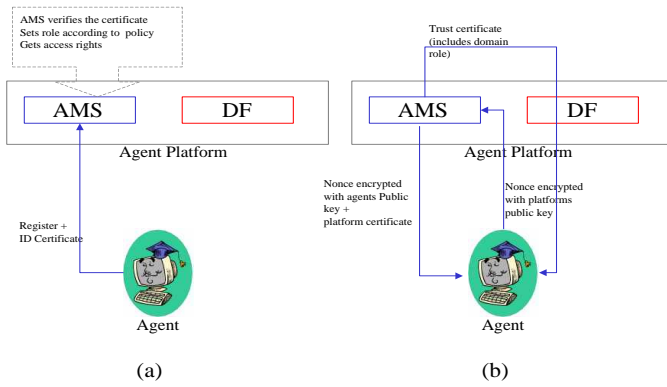


Figure 1: This images represents the secure registration of an agent with the AMS on the agent platform. (a) To start the registration process, an agent sends a signed registration message along with its ID certificate to the AMS. The AMS verifies the ID certificate based on its verification rules, and checks the signature. It then assigns roles to the agent and decides the access rights of the agent. (b) The AMS starts the PKI handshaking protocol and sends the platform's certificate along with a nonce encrypted with the agent's public key to the address specified by the agent. The agent decrypts the nonce and sends it back encrypted with the platform's public key. Once the handshaking is complete, the AMS sends the agent a trust certificate which includes its roles in the platform. The AMS associates the trust certificate with the agents public key.

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 temporary role of the agent, the associated public key 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.

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.

Figure One demonstrates the steps involved in registering with an AMS.

6.2 Security Module for a DF

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 DFs policy allows that particular agent or role to register. 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, 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. DF checks the role based rights of the agent, if the agent has the right 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.

Figure Two illustrates the steps involved in registering with a DF.

6.3 Security Module for Agent

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. So after the requests are filtered by the DF, they can be re-verified by the service agent.

After an agent receives some services as a result of its DF query, it can go ahead and try to execute the service. The agent sends a request to agent offering the service 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 assign a role to the agent, and checks 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 imposed by the service agent, that the requesting agent fails to meet. An agent's security module operates on a local policy which is made up of rules for verification of trust certificates, authentication, role assignment, and access control specific to the service agent's domain.

6.4 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 cer-

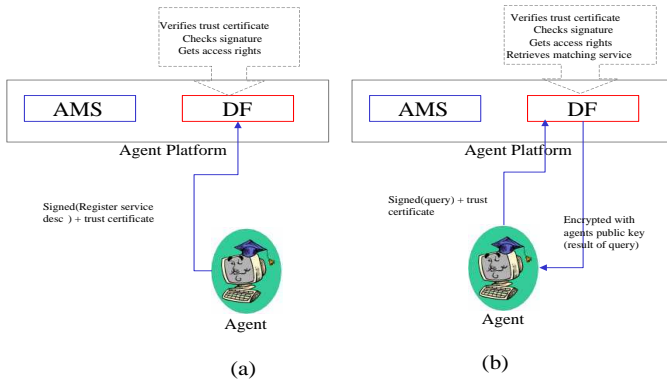


Figure 2: This images represents the secure registration of an agent with the AMS on the agent platform. (a) The agent sends a signed service registration request to the DF and attaches its trust certificate. The DF checks the certificate to make sure it is still valid and actually belongs to the agent. The DF retrieves the public key associated with the trust certificate to check the signature on the message and then checks the access rights of the agent and the agent's roles. If the agent is allowed to register services, the DF proceeds with the registers. (b) To query the DF, the agent sends a signed query and its trust certificate. As before, the DF checks the certificate and signature before proceeding with the searching. It then retrieves a list of matching services, whose conditions are fulfilled by the agents credentials. This list is sent back to the agent encrypted with the agents public key.

tificate, which contains information about its roles and 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. So an agent could have different roles on different platforms, and so rights vary according to the platform as well.

DF's of different platforms can be accessed if they register with each other through principles of *federations of DF* [9]. 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.

7. EXTENSION OF RBAC

Simple Role Based Access Control is rather restrictive, as every time access rights of an agent change, its role has to change as well. Our work is similar to role based access control [20] - an approach in which access decisions are based on the roles that individual users have as part of an organization, such as doctor, nurse, manager, or student-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 an XML based 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.

Our framework allows certain authorized agents to delegate access rights, with restrictions attached, to other agents. But rights can only be delegated by agents who have the right to delegate. 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. As per our Distributed Trust Model, Section 3, repeated re-delegations form a delegation chain. We have defined an ontology for delegation along with a set of rules that enforce all the constraints of the delegations. To delegate a right to another agent or set of agents, the delegator either creates a short-lived delegation certificate for the delegatee or sends this delegation information to the AMS and DF of the platform on which the right exists (could be right to use some AMS or DF service or right to use some agent service), or sends this delegation to the agent offering the service. A short-lived certificate is a certificate that is valid only for a short period of time to allow for faster revocations. When an agent in the delegated group comes along, it either hands over the delegation certificate or the access control rules trigger off the delegation sent by the delegator, and the agent now can perform the delegated right. If the delegation information is sent to the service agent, it is up to the service agent to validate and honor the delegation. The security agent should change its entry in the DF to reflect the delegation. But these delegations and the delegation certificates are valid for a period proportionate to the amount of trust the delegator has in the delegatee. Once the delegation expires, the delegatee no longer has the right. We have defined different types of delegations based what rights they confer.⁰

Revocation also raises some interesting issues. What happens if an agents identity certificate is revoked for some reason? The trust certificate is valid for a period proportionate to the trust the platform has in the agent or the agent's organization or agent's CA. So, for example, if the agent's organization is extremely trustworthy and so is the CA, the trust certificate would be valid for a longer time, than if the agent's organization was unknown and the CA was somewhat trusted. Once this trust certificate expires, the agent must approach the AMS for a new certificate. During this process, the AMS may decide to verify the identity certificate with the CA based on the policy. If the identity certificate is revoked, then the AMS cannot validate the agent and the request fails.

Another problem occurs when a delegated right is revoked, or if an agent changes its policy and does not want a certain

⁰There are several issues that arise when the agent offering the delegated service moves to a new platform. Some possibilities are that the agent queries the platform for any delegated rights on its services and gives them to the new platform, or the old platform could send them directly to the new platform, or the delegator could delegate the rights again on the new platform.

role or set of agents to access its service. A delegation could either be in the form of a certificate or a message to the involved platform or service agent. If a delegated right is revoked, then the platform on which the delegation was made, or the service agent has to be informed. As the delegator agent sent the delegation in the first place, it could probably send the revocation as well. This revocation causes the delegation rules to be blocked and the agent is denied the rights immediately. If it was a delegation certificate, the certificate would have to time out, before the revocation came into place. Since the time of the delegation certificate depends on the trust level of the delegatee, if the agent is not to be trusted, the certificate will be valid for a short time and the revocations will be effective fast. If the conditions of the services of a service agent change, the service agent should inform the DF about the changed constraints or the changed categories of the services.

Prohibitions are also part of the framework. A prohibition prevents an agent from accessing a certain service. It is incorporated as a permanent revocation. We have not worked with releasing prohibitions, but view it as an extension of delegating a permission.

As with delegation, not every agent can revoke rights. Only authorized agents with the right to revoke can actually revoke. The right to revoke is implicit in this system. An agent can revoke an right that it has delegated and an agent can revoke any rights on its own services.

To complement delegation, we allow *requests for permission* as well. An agent can ask another agent to delegate to it the right to access a certain service. If the receiving agent is satisfied with the requester's credentials, it will either send back a delegation certificate or inform the correct platform or service agent.

8. DELEGATIONS

Only agents with the ability to delegate can make valid 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 authorized delegations actually change access rights of other agents, the others are voided. The right to delegate is defined implicitly and explicitly. Implicitly, an agent can delegate rights to any service it offers. But explicitly, an agent that has been given the right to delegate by another agent, who has the right, 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 makes sure of this, before allowing an agent to access a service. Revocation in a delegation chain causes all the delegations after the revocation to be invalidated.

As delegation is so vital for security in a multi platform system, we describe the process in detail in this section. A *delegation* is the transfer of rights from one agent to another. To avoid problems with insecure delegations, our framework

supports the addition of constraints to delegations. 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. To restrict 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, so that a right to delegate can be delegated without giving the right to actually use the right.

To explain the power of these delegations, we classify delegations into the following five categories: simple, group, restricted execution, re-delegation and strict re-delegation.

In a *Simple delegation*, an agent is given the right to execute a certain service on a platform or agent; there are no constraints on delegatee or execution and the delegatee cannot redelegate. Agent A delegates to agent B the right to use one of its services for a certain period without giving B the right to further delegate.

A *Group delegation* is an extension of a simple delegation, but it is made to a group that satisfies the constraints on delegatee. An agent A delegates the right to some service to anyone who is registered with a certain platform, and with certificates from a certain trusted CA, but not the right to redelegate. The constraint on re-delegation is set to false, and all the other restrictions are part of the constraints on delegatee. The constraint on execution is set to true.

A *Restricted Execution* could either be a simple delegation or a group delegation with additional restrictions on execution. The constraints on execution is set to the conditions the delegatee must satisfy in order to use the right, and the constraints on re-delegation is set to false. An agent A delegates to a group some right, but only agents called John can actually execute this right.

A *Re-delegation* could be any one of simple, group or restricted execution with constraints on re-delegation. This specify the conditions that a delegatee should satisfy to be able to re-delegate this right. An agent A delegates the an agent B, the right to execute some service and the right to re-delegate this right to anyone registered on the same platform. Agent B delegates this right to agent C which is not registered at agent B's platform. When agent C tried to execute service, the delegation mechanism catches the invalid delegation and the access fails. Agent B delegates

A *Strict re-delegation* is re-delegation with its constraint on execution set to false. This means that the delegatee can re-delegate but not execute this right.

9. ONTOLOGIES

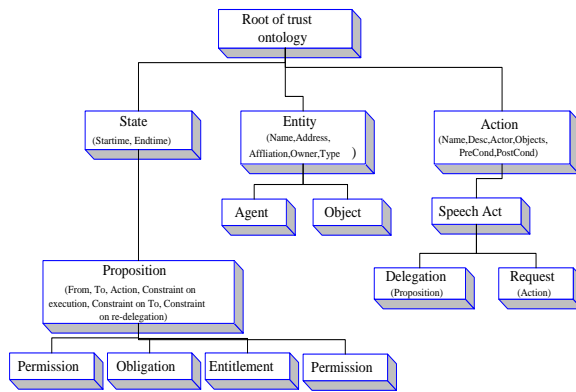


Figure 3: Our trust ontology as a class hierarchy, with the properties associated with each class in brackets.

Our infrastructure uses several ontologies, expressed in DAML+OIL¹ to represent security information and policies in a multi-agent system.

We have an ontology for trust and security information in this system, which is illustrated in Figure Three. 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 subclassified into Permission, Obligation, Entitlement and Prohibition. 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 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 Four shows a small portion of the ontology.

10. POLICY

The security policies are based on the Agent System ontology. Each platform and agent follows a security policy. A security policy may contain rules for verifying certificates, authentication, role assignment, access control, delegation, revocation and prohibition. A policy also contains a role based access control list, for generic rights associated with roles. 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. Role assignment rules will list the conditions an agent must satisfy to be of a certain role. Rules for access control will mention 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 dele-

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
  xmlns:rdfs="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:df="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:trust="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-rdf-literal</daml:equivalentTo>
    </daml:Class>
    <daml:Ontology>

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

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

    <!-- SubClass of Certificates; ID, Trust, Delegation -->
    <rdfs:Class rdf:ID="IDCertificate">
      <rdfs:subClassOf rdfs:resource="#Certificate"/>
      <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 rdfs:resource="#Certificate"/>
      <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 rdfs:resource="#Certificate"/>
      <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 rdfs:resource="#trust:Object"/>
    </rdf:Property>
    <rdf:Property rdf:ID="PublicKey">
      <rdfs:domain rdfs:resource="#trust:Object"/>
    </rdf:Property>
    <rdf:Property rdf:ID="StartDateTime">
      <rdfs:domain rdfs:resource="#Date"/>
    </rdf:Property>
    <rdf:Property rdf:ID="EndDateTime">
      <rdfs:domain rdfs:resource="#Date"/>
    </rdf:Property>
    <!-- more properties ... -->

    <!-- Subclass of Object -->
    <rdfs:Class rdf:ID="Organization">
      <rdfs:subClassOf rdfs:resource="#trust: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 rdfs:resource="#trust:Action"/>
      <rdfs:label>RegisterWithAMS</rdfs:label>
      <daml:Restriction>
        <daml:onProperty rdfs:resource="#trust:Actor"/>
        <daml:toClass rdfs:resource="#trust:Agent"/>
      </daml:Restriction>
    </rdfs:Class>

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

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

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

Figure 4: 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.

gated 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. Also, an agent can only access a service if it has the role based access right, or the right has been delegated to it *and the agent is not prohibited from using that service*. This policy can be overridden by a less strict policy.

11. EXAMPLE

We describe an example scenario to further emphasize the functionality of the framework. Consider 2 personal agents i.e. agents that are representations of users, agent Alice from ABC Consultants and agent Bob from XYZ Consultants. There also exists agent Carol, a service agent i.e. an agent that provides some sort of service. Alice and Bob both register on different platforms according to Figure One. Carol registers with another platform in a similar fashion, and then registers her service, a dating service, according to

Figure Two. She classifies her service as 'secure' and only allows personnel of ABC Consultants to access it, but gives them the right to delegate this access right to anyone they trust. But she does not allow them to delegate the right to delegate this right. Alice is assigned the role of 'manager' and has all the access rights associated with 'manager' on that domain. Similarly, Bob is a 'programmer', whereas Carol is assigned the role of 'guest' on their respective platforms. Carol's DF registers with Alice's DF and Bob's DF according to the principles of DF federations. Now Alice can search through services on her platform as well as Carol's platform. She searches for a dating service. Her DF sends her information to Carol's DF. Carol's DF checks Alice's trust certificate and her credentials. It then searches for a dating service and comes up with Carol's. The DF then checks the category and then makes sure that Alice satisfies Carol's conditions on the service. Then the DF returns Carol's address to Alice. Alice uses Carol's service and finds it extremely useful. She tells Bob about it. But when he tries to find it, he can't. He asks Alice if she can give him permission to access this right. As Alice trusts Bob, she gives him the right to use the dating service and the right to delegate this service. She sends him a delegation certificate. Bob queries his DF again with the delegation certificate and his ID certificate. His DF forwards this message Carol's platform. This time Carol's platform has information about the delegation. Carol's DF verifies the delegation certificate and the ID certificate. Carol's conditions allow Bob access to the dating service. Bob is also very happy with the service and decides to allow his friends at XYZ Consultants to access the service as well. He delegates the access right to all his friends, because Alice allowed him to delegate. But when his friends try to access the service, they are denied access because Carol's condition invalidates Bob's delegation.

12. SUMMARY

We have described an approach to some security problems in multiagent systems based on distributed trust and the delegation of permissions, obligations and credibility. This approach is particularly useful in open environment in which agents must interact with other agents with which they are not familiar. We have used this approach in two implemented systems – an agent-based supply chain management application and an agent-mediated pervasive computing environment. Finally we present a design that provides security functions. We are currently adapting it for use in the FIPA agent framework using DAML+OIL as a language in which to encode policies, roles, permissions and obligations.

There are a number of extensions and enhancements which we are beginning to explore. We are studying the feasibility of integrating SEMi-trusted Mediator (SEM) model [4] because of the large amount of encryption and decryption required in this system, which would be computationally difficult on agents residing on mobile devices. We are currently examining the feasibility of developing rules and constraints in DAML+OIL. Though our ontologies are encoded in DAML+OIL, we still use horn clauses in Prolog to specify rules for verification of certificates, rules for delegation propagation etc. We are considering *distributed belief* as a way for the agent to garner the required trust information. The policy could include rules for belief as well. For example, the AMS would believe that Mark had the role of Manager

in XYZ, if two registered and trusted agents say that it is true.

Research in security for agent systems area often tends to focus on a limited subset of the security challenges in MAS, like distributed access control, distributed trust, etc. We believe our security model to be a good attempt at developing an *entire framework* that tries to solve most of the security issues in these environments. It is also easily adaptable to any particular domain because we have designed an ontology that can be extended to work with domain specific information.

13. REFERENCES

- [1] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.
- [2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The keynote trust management system version, 1999.
- [3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems, 1999.
- [4] D. Boneh, X. Ding, G. Tsudik, and B. Wong. Fast revocation of security capabilities. In *Usenix Security Symposium, 2001*, 2001.
- [5] C. Castelfranchi and R. Falcone. Principles of trust for mas: Cognitive anatomy, 1998.
- [6] DAML. Daml specification (<http://www.daml.org/2001/03/daml+oil-index>), 2001.
- [7] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, Maryland, 1994. ACM Press.
- [8] T. Finin, Y. Labrou, and Y. Peng. Mobile agents can benefit from standards efforts in inter-agent communication, 1998.
- [9] FIPA. Fipa agent management specification. In *FIPA website* (<http://www.fipa.org/specs/fipa00023/>), 2001.
- [10] Qi He, Katia Syraça, and Tim Finin. Personal security agent : Kqml-based pki. In *In Proceedings of the Second International Conference of Autonomous Agent, May 1998*, 1998.
- [11] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid. Access control meets public key infrastructure : Or assigning roles to strangers. In *2000 IEEE Symposium on Security and Privacy, Oakland, May 2000*, 2000.
- [12] Yuh-Jong Hu. Some thoughts on agent trust and delegation. In *Autonomous Agents 2001*.

- [13] IETF. Simple public key infrastructure (spki) charter: <http://www.ietf.org/html.charters/spkicharter.html>.
- [14] NJ) Ingersoll Rand (Woodcliff Lake, CA) QAD (Carpenteria, IL) Berclain Group (Schaumburg, and NY) IBM Corporation (Somers. Ciimplex consortium, consortium for integrated intelligent manufacturing planning and execution, 2000.
- [15] Lalana Kagal, Tim Finin, and Anupam Joshi. Trust based security for pervasive computing environments. In *To appear in IEEE Communications, December 2001*, 2001.
- [16] Lalana Kagal, Tim Finin, and Yun Peng. A framework for distributed trust management. In *To appear in proceedings of IJCAI-01 Workshop on Autonomy, Delegation and Control*, 2001.
- [17] Lalana Kagal, Jeffrey Undercoffer, Tim Finin, and Anupam Joshi. Vigil : Enforcing security for pervasive environments. In *Under review*, 2001.
- [18] Ninghui Li. Delegation logic: A logic-based approach to distributed authorization.
- [19] Ninghui Li, Benjamin N. Grosz, and Joan Feigenbaum. A practically implementable and tractable delegation logic. In *In Proceedings of IEEE Symp. on Security and Privacy, held Oakland, CA, USA, May 2000*, 2000.
- [20] Emil C. Lupu and Morris Sloman. Towards a role based framework for distributed systems management. *Journal of Networks and Systems Management*, Plenum Press, 1996.
- [21] 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*, 200.
- [22] M.Blaze, J.Feigenbaum, and J.Lacy. Decentralized trust management. *IEEE Proceedings of the 17th Symposium*.
- [23] M.Blaze, J.Feigenbaum, and M.Stauss. Compliance checking in the policy maker trust management system. In *Proceedings of Financial Crypto'98, Lecture Notes in Computer Sciences vol.1465, Springer Berlin, 1998*, 1998.
- [24] Agostino Poggi, Giovanni Rimassa, and Michele Tomaiuolo. Multi-user and security support for multi-agent systems. In *Proceedings of WOA 2001 Workshop, Modena, Sep 2001*, 2001.
- [25] C. Thirunavukkarasu, T. Finin, and J. Mayfield. Secret agents – A security architecture for the KQML agent communication language. In *Intelligent Information Agents Workshop held in conjunction with Fourth International Conference on Information and Knowledge Management CIKM'95*, Baltimore, 1995.
- [26] 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.
- [27] N. Yiaelis, E. Lupu, and M. Sloman. Role-based security for distributed object systems, 1996.
- [28] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.