A Delegation Based Model for Distributed Trust

Lalana Kagal, Timothy Finin, Yun Peng
Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250
email : {lkagal1,finin,ypeng}@cs.umbc.edu
phone : 410-455-3971
fax : 410-455-3969

Abstract

In this paper we outline an infrastructure that facilitates security and trust management in a multi-agent system. Our model eases the problem of authorization in a network of heterogeneous agents and also contains mechanisms for delegation of authorization information. The framework allows agents to exchange trust information using a series of Interaction Protocols based on FIPA (Foundation for Intelligent Physical Agents) Interaction Protocols (FIPA 1998). It decentralizes security decisions, enabling more than one agent to be responsible for the validation of requests or for the delegation of permissions. It is very flexible and encourages mobility because the process of requesting services and granting access is divided into two independent steps. This allows an agent to disconnect after the first step and reconnect elsewhere to continue the process of securing the service. The model also uses a policy based approach, to specify rules for authorization and delegation, and a distributed knowledge base, that contains information about the interacting agents. We describe an implemented system that incorporates our framework using X.509 certificates and a Prolog knowledge base.

Keywords
Authorization, security, distributed trust, agents, X.509 certificates, knowledge representation, role based, policy

Introduction

Authorization in a distributed system is quite different from a centralized system. Traditionally, authorization is composed of authentication and access control. The Access Control List (ACL) (J.K.Jan 1991; M.S.Hwang 1994), which works by attaching access control information of the subject to the resource, is a popular approach. Role based access control (Blaze, Feigenbaum, & Keromytis 1999; Sandhu *et al.* 1996) is another scheme that has been widely used. In this model, the the large number of users forces a division into groups (or roles) and the access information is attached to roles. However both these schemes are unable to meet the requirements of a large distributed system because the individuals that need to access the resource may not be known ahead of time, so the ACL or role based information cannot be formed. Also in a business environment, people are frequently changing jobs or positions and roles.

We try to solve this problem through the application of a chain of trust using rights and delegations. In this system, we model permissions as the rights of an agent. We associate rights with actions, so possession of a right permits the corresponding agent to perform a certain action. We are currently exploring the use of obligations and the repercussions of failing to fulfill obligations.

Rights or privileges can be given to trusted agents that are then responsible for agents they may delegate this 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 the failure are not permitted to perform the action associated with the right.

This paper is organized as follows : *Related Work* discusses other similar approaches, and *The Problem* discusses the problem and the two scenarios we target. We describe our infrastructure briefly in *Infrastructure*, the knowledge base is described in *Knowledge Base* and the policy being used in *Policy*. *Ontology* explains our ontology and *Interaction Protocol* discusses our protocols for agent communication. We explain the software used for each component in *Implementation Details*. *Future Work* is a discussion of future research directions, and *Conclusion* contains the summary of our work.

Related Work

Blaze, who coined the term Distributed Trust Management, tries to solve the problem by the access checking method, but without any authentication (M.Blaze 1996; Blaze *et al.* 1998). The Simple Public Key Infrastructure (SPKI) was the first proposed standard for distributed trust management (Ellison *et al.* 1998). This solution, though simple and elegant, does not help in delegations. W. Johnston's Use-Condition Centered Approach (Johnston & Larsen 1996) uses certificates for use-conditions that are created by those responsible for the resources. This can only be used when the resource is simple enough to

To appear in Proceedings of The IJCAI-01 Workshop on Autonomy, Delegation, and Control.

This research was supported in part by the IBM EECOMS program and the DARPA DAML program under contract F30602-97-1-0215

be described by use-conditions, but in large systems there could be many types of access like read, write, execute etc. Another trust management system is TE (Herzberg *et al.* 2000) from IBM. This s not able to address all the relevant issues because it considers only role authorization. Delegation logics (Li, Feigenbaum, & Grosof 1999; Grosof & Labrou 1999), from IBM, is very similar to our approach, however is not able to capture adequately the constraints associated with rights and delegations.

The above mentioned models are very powerful, however they do not meet all the requirements of trust management. Generally security systems should not only authenticate users, but also allow users to delegate their rights and beliefs to other users securely and provide a flexible mechanism for this delegation. The above systems either support only authentication ignoring delegation altogether, or support delegation to some extent without providing the flexibility needed, or do not provide sufficient restrictions on delegation of rights.

We drew on the key points of most of the above-mentioned schemes and designed an infrastructure that uses X.509 certificates and policies to enforce security. A policy contains basic/axiomatic rights, rights associated with roles, rules for delegation, and rules for checking the validity of requests. X.509 certificates are used not only for identity purposes, but also for authorization and delegation. Our system allows agents to delegate any right that they may have. Whether these delegations are honored depends on the policy. Constraints can be added to both the actual delegation and to the delegatee, tightening control on the rights and permissions. In our model, we use a 'redelegatable' flag that controls whether the right can be further delegated. We have found that these features of our system address the main issues of trust management, authentication and delegation, successfully.

The Problem

We have tried to solve the problem of authorization and delegation in a system that consists of widely distributed resources and agents. There are two scenarios that we have tackled; a home/office automation model and an electronic supply chain management system like EECOMS (Ingersoll Rand 2000). We have been successful in implementing the EECOMS scenario and are currently working on an office automation scenario with BlueTooth (Bluetoothwebsite 2001) enabled devices.

EECOMS : IBM project

This work is sponsored by the CIIMPLEX consortium (Ingersoll Rand 2000) 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.

The EECOMS project deals with trust establishment in a supply chain management system. Generally, buyers and suppliers need to share certain information with each other. Our system sets up authorization and delegation rules, so that this information may be accessed only by those authorized to do so. Special intelligent agents called 'security agents' are required for authentication and authorization within a particular domain, and are trusted within and outside the group/company. 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.

The 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 (refer to *Policy*). These security agents enforce the security policy of the company. The policy is not changed frequently and usually involves human intervention. To expedite the identification of each agent, we assume that every agent has an Identity Certificate (ID) issued by a trusted Certificate Authority (CA).

Home/Office Automation

Our architecture could apply to the wireless world in the following scenario. If a visiting lecturer at a University needs to use a projector in a lecture hall, she/he needs to be delegated the right by some authorized personnel. If the policy states that all professors can use the projector and that professors can delegate this right to the lecturer, the lecturer can obtain the 'token' from a professor. Using a hand-held device such as a PDA, mobile phone etc. the visitor beams her/his identifying token to the projector along with the delegation token. The projector may or may not have the processing power to reason about these certificates and rights. If it does not have the capability, the agent in the projector sends the token (using wireless or wire line communication) to a 'smart' agent that evaluates the request and returns the result. The agent that does the reasoning needs to check the identity of the requester and then make sure that the requester has the right to access the projector. In this case, the requester has been delegated the right by a professor, so the agent should verify that the professor has the right to delegate. Once the request is validated, the visitor can beam her/his slides to the projector agent that starts up the presentation.

We have started experimenting with Bluetooth (Bluetoothwebsite 2001) and believe that the above scenario is not too far in the future (Lalana Kagal 2001; Chen & Chakraborty 2001).

Infrastructure

Our architecture assumes that each group of agents, known as policy domains, is protected by one or more *security agents*. These agents are responsible for authorizing access to services/resources within that group. These agents access the policy and knowledge base associated with the domain. The knowledge base, encoded in Prolog, contains information about the agents in the domain, including their name, role/position, age and other characteristics associated with an employee . All delegations are stored with the security agents, which have the ability to reason about them. An agent (requester) can execute a right or access a resource by providing it's identity information to the security agent. The security agent checks this information for validity and reads its policies to verify that the requester has the right. If the requesting agent does not have the right, the security agent returns an error message. Otherwise it forwards the request to the agent in charge of the resource, the accessor agent, along with a message indicating that the request is authorized by the security agent. As the security agent is trusted by every other agent in the system, the requesting agent is granted access. If the accessor agent has the computing power to reason about certificates, rights and delegations the request can be sent directly to it, instead of via the security agent.

The requester can also obtain access to a certain resource that it previously could not access, through a delegation from an authorized agent (delegator). An authorized agent (an agent with the ability to delegate a certain right) delegates the right by sending a message to the security agent. The delegation has to be approved by the security agent and should conform to its policies. The requester approaches the security agent with its identity information and a request for permission. The security agent verifies the identity of the requester and checks with its policies to make sure that the requester can be given access to the resource. The new delegation makes the request valid. The security agent generates an authorization ticket/certificate which contains a Prolog (Swedish Institute 2001b) like statement giving the requester permission to access the resource. This message is sent to the requesting agent. These statements are dated and are valid only for a certain period of time. While the statements are valid, the requesting agent can use them as tickets to access the resource. This allows the entire process of verification and reasoning to be skipped, and the requester gets access to the resource as soon as the authorizing statement is recognized and verified by the accessor agent.

All the reasoning about rights and delegations is handled by a set of Prolog rules causing incorrect delegations and statements to be trapped by Prolog's backward chaining mechanism and prevented from going through. We have rules that cause constraints on rights to be propagated when a delegation occurs.

A delegatee is an agent that delegates a certain right to another agent or group of agents. It has the permission to perform a certain action and also the ability to further delegate this right. A delegatee will only delegate to an agent that it trusts since the delegatee is held responsible for the actions of the agents it has delegated to.

We use X.509 certificates for identity certificates and for encapsulating delegations. The ITU-T Recommendation, X.509, has been implemented as a de facto standard. X.509 focuses on defining a mechanism by which information can be made available in a secure way to a third-party. X.509v3 specifications define a certificate as: *user certificate; public key certificate; certificate: The public keys of a user, together with some other information, rendered unforgeable by encipherment with the private key of the certification authority which issued it.*

Knowledge Base

We use a number of predicates to represent the information flowing in the system. We also describe the agents in the policy domain and store this information in a knowledge base. A security agent uses this knowledge along with the policy while granting permission to an agent. We encode in Prolog details about the name, role, etc. of the agent. All information the system learns is also added to the knowledge base; delegations and requests.

Request

An agent requests a security agent to perform some action on his behalf. The security agent will perform this action only if the agent has the ability to do so.

```
request(<fromAgent>,<action>)
```

An agent can also request for permission to perform a certain action. If the requested agent is satisfied with the agent's credentials, this request will result in a delegation from the requestee to the requestor.

```
requestPermission(<fromAgent>,<action>)
```

Delegation

An agent can execute any action that it has the right to execute, or has been has been delegated the right to execute. It can also delegate this right to other agents, if it has been authorized to subsequently delegate. The agent can also delegate all the axiomatic rights that it possesses. A delegation itself is a right which can be delegated. In other words, an agent could be given the ability to perform some action but not to further delegate it, given the right to some action and the permission to delegate it, or the ability to delegate some action but not the ability to execute it.

So, an agent can delegate any 'delegatable' right. This leads to a chain of delegation, and if any one link is no longer valid the access is denied. We also allow for constraints on rights, delegations and ability to re-delegate.

One of the main features in our system is that false delegations are not rejected as soon as they enter the system, but are stored for later evaluation of a possible security breach. 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 in the delegation chain.

The statement that is used to describe delegations and constraints on delegations is :

```
delegate(IssueTime, StartTime, EndTime,
 From, To, canDo(X, Action, CDC),
         IDC, Redelegatable)
```

- IssueTime : when the statement was issued
- StartTime : when the delegation becomes valid
- EndTime : when the delegation becomes invalid
- From : delegator agent
- To : delegatee agent
- canDo(X,Action,CDC) : delegated action, X has the right to the action, only if X satisfies the condition CDC
- IDC : condition on the delegation
- Redelegatable : true if the delegator, To, has the permission to redelegate the action

Types of delegations Our work in the EECOMS scenario involved several different types of delegations which we describe here and give simple examples.

- Time Bound Delegation : It is a delegation that is valid only for a certain time period

  ```
  delegate(1105001120,1105001121, 1110001120,
           From, X, canDo(Y, Action, CDC),
           employee(X,abc), Flag)
  ```

  This delegation is only valid between 1105001121 and 1110001120.
- Group Delegation : It can be used to delegate rights to a group of agents who satisfy certain conditions

  ```
  delegate(IssueTime, StartTime, EndTime,
           From, X, canDo(Y, Action, CDC),
           (employee(X,abc),age(X,24)), Flag)
  ```

  This delegates the right to perform *Action* to a group of employees of *abc* who are 24 years old.
- Action Restricted Delegation : This forces the delegatee to satisfy certain conditions before the action can be carried out

  ```
  delegate(IssueTime, StartTime, EndTime,
           From, X,
           canDo(Y, Action, name(Y,john) ),
           (employee(X,abc),age(X,24)), Flag)
  ```

  Only employees of *abc* who are 24 and named *john* can execute this action, though all employees aged 24 have been delegated the right.
- Redelegatable Delegation : In this delegation, a right can be delegated along with the right to re-delegate the right.

  ```
  delegate(IssueTime, StartTime, EndTime,
           From, To, canDo(X, Action, CDC),
           IDC, true)
  ```

  This statement allows the recipient to further delegate the right.
- Strictly Redelegatable Delegation : This statement allows a right to be re-delegated without giving the delegatee the right to actually do the action.

  ```
  delegate(IssueTime, StartTime, EndTime,
           From, john,
           canDo(Y, Action, notname(Y,john)),
           IDC, true)
  ```

  john is given the right to further delegate the action, *Action*, but not the permission to execute the action himself.

Policy

Each domain has a policy associated with it (Lupu & Sloman 1997; Lupu *et al.* 1995). This policy consists of authorization policies and delegation policies. Authorization policies deal with the rules for checking the validity of requests for actions. An example of a rule for authorization would be checking the identity certificate of an agent and verifying that the agent has an axiomatic right. Delegation policies describe rules for delegation of rights. A rule for delegation would be checking that an agent has the ability to delegate before allowing the delegation to be approved. A policy also contains basic or axiomatic rights, and rights associated with roles. We introduce the concept of primitive or axiomatic rights, which are rights that all individuals possess and that are stored in the global policy. For example, every citizen of India has the right to vote, and anyone who owns a database has the right to delegate the right to read from/write to that database. These are basic rights that are not often expressed, but used implicitly. All policies are described in Prolog. A policy can be viewed as a set of rules for a particular domain that defines what permissions a user has and what permissions she/he can obtain.

Users of the system are generally assigned roles. A role is defined as a collection of rights and duties (Sandhu *et al.* 1996; Lupu & Sloman 1997; Lupu *et al.* 1995). Roles are arranged in a hierarchy, so that rights can be inherited. An entity has a right if it is mentioned in the policy or if the right has been delegated to it by another entity that has the ability to delegate. Delegations generally flow downwards in the role hierarchy, and are from a higher role to a lower role. However our framework does not strictly adhere to role based access, and allows rights and delegations to be assigned to individuals and groups. This overcomes the drawbacks of Access Control Lists and Role Based Access.

Rights As rights are used throughout the system, we describe the syntax in more detail. In our system, we model permissions as rights that an agent possesses. We associate rights with actions, so a right implies that the corresponding agent is permitted to perform a certain action.

Our system encodes rights into a logical form in Prolog as the following :

```
rightToDo(agentName, Action, Constraint)
```

- agentName : URI for the agent
- Action : representation of the ability eg. accessDB(db5,read)
- Constraint : restriction on the right, eg. employee(agentName,XYZ)

Using this statement, all kinds of permissions on actions can be specified. An agent is given the right to do a certain action based on a constraint. An agent can execute the action only if it satisfies all the constraints.

Ontology

Our approach uses a simple ontology of agents, propositions and actions which are briefly described below.

- Agents : An agent is an entity in the system, which could be a program or a human.
- Propositions : We use two propositions, ability and delegate :
  - Ability is a property that an agent has. An ability is true if an agent has the right to perform the action.

    ```
    canDo(<agent>,<action>,<constraintsOnAction>)
    ```
  - Delegate is a proposition asserted into a database saying that one agent delegates to another agent the right to perform some action.

    ```
    delegate(<issueTime>,<startTime>,<endTime>,
             <fromAgent>,<toAgent>,<ability>,
             <constraintOnDelegation>,
             <redelegateFlag>)
    ```
- Action is what an agent can perform and is closely linked to abilities.

  ```
  accessDB(db5,read)
  ```

Interaction Protocols

We have developed a set of Interaction Protocols based on FIPA (FIPA 1998) for communication between the agents in the domain. Each agent communication is an object known as Signed Message Object (SMO). An SMO consists of a list of certificates, the request or the authorization statement signed with the senders private key, and other required fields. The relevant certificates are included as part of the communication data structure to expedite the authorization process. The SMO contains two text fields, *msg* and *signedMsg*. For example, if an agent after acquiring a delegation for a particular right, wished to perform the action, it would send a request SMO to the agent controlling the resource. The SMO would contain the requesting agent's identity certificate and the delegation certificate. The clear text request for access would be in the *msg* field of the SMO and the *signedMsg* field would consist of the signed request for access. The receiving agent would verify the certificates and check the signed request against the clear text request. If all the checks went through, the request would be permitted.

Prolog Predicates

The Interaction Protocols use certain prolog predicates, embedded into SMOs, that are described below :

- An agent requests another agent to perform some action on his behalf. The latter agent will perform this action only if the former agent has the ability.

  ```
  request(<fromAgent>,<action>)
  ```
- An agent requests permission from another agent who has the ability to delegate. This results in an error or a delegation depending on the credentials of both the requestor and the requestee.

  ```
  requestPermission(<fromAgent>,<action>)
  ```
- An agent can ask a security agent if it has the right to perform the action. This results in a *tell* with the proposition being a *canDo* (refer to *Ontology*).

  ```
  ask(<fromAgent>,<toAgent>,<action>)
  ```
- Idelegate is the action of delegating the ability to perform the action from one agent to another.

  ```
  idelegate(<startTime>,<endTime>,<fromAgent>,
            <toAgent>,<ability>,
            <constraintOnDelegation>,
            <redelegateFlag>)
  ```
- An agent can tell another agent a proposition that it believes is true.

  ```
  tell(<fromAgent>,<toAgent>,<proposition>)
  ```

Example

Let us assume that there are two organizations, ABC and XYZ, that are collaborating on a certain project. If a Software Consultant (SC) working with ABC needs to access a database of her/his client, XYZ, she/he first needs to get the correct authorization from her/his supervisor. Let us assume that the supervisor has the right to access the database (a rightToDo refer to *Rights*) and that the right can be delegated. So the supervisor sends a certificate with the following content containing a delegate to SC . SC then uses this certificate to create a SMO and sends this SMO to the security agent of ABC. Before the authorization is given at ABC the security agent will check all SC's credentials by looking at its policy and the SMO. It checks if SC is working for ABC, whether XYZ is indeed her/his client etc. This is required because otherwise it could lead to a breach in security. The security agent returns an authorization certificate. SC needs to show this authorization certificate, along with her/his other certificates, to the security agent at XYZ. The security agent at XYZ will double check all the credentials, making sure that the security agent of ABC is trusted and SC indeed has the right to access the database. It will create a certificate and send it back to the SC. SC can now use this as a 'ticket' to access the database.

Protocols

Request for Action An agent requesting a certain action of another agent outside the company, creates a SMO with its ID certificate and sends it to the security officer along with other certificates that strengthen his case. The security officer checks the credentials supplied by the requester permitting the request to go through only if all SMOs are valid. If it is an inter-company information request, the request is sent to the security officer of the recipient's company. There the request and the attached credentials are verified once again and then forwarded to the agent controlling access to the information with an additional attachment reconfirming the authenticity of the request. For actions on an agent within the company such a high degree of security may not be necessary. The recipient, if intelligent, can validate the SMO and reason whether the action should be allowed. Otherwise the recipient could ask the security officer to process the message for it.

We illustrate the working of Request for Action by an example. ABC and XYZ are two companies represented by their security agents, SA-ABC and SA-XYZ. XYZ is the client of ABC. Marty is a design engineer in ABC, where design engineers can access their client's database, db5. The following steps illustrate how the authorization actually takes place.

1. SA-XYZ accesses the company policy for XYZ and the global shared policy
2. SA-ABC accesses the company policy for ABC and the global shared policy
3. SA-XYZ sends a message to SA-ABC saying that SA-ABC has the right to delegate access to db5, which is a database in XYZ, to all employees.

   ```
   tell(sa-xyz, sa-abc,
       idelegate(StartTime, EndTime, sa-xyz,
                 sa-abc, canDo(X,accessDB(db5),
                 employee(X,abc)), true,true))
   ```

   SA-ABC asserts the proposition

   ```
   delegate(IssueTime, StartTime, EndTime, sa-xyz,
            sa-abc, canDo(X,accessDB(db5),
            employee(X,abc)),true,true)
   ```

   SA-ABC gives all Design Engineers the right to access db5, but not the ability to delegate.

   ```
   tell(sa-abc,sa-abc,
        idelegate(StartTime, EndTime, sa-abc,
          X, canDo(X, accessDB(db5),true),
          role(X,designEngineer),false))
   ```

   This causes a delegate statement to be inserted into the knowledge base.

   ```
   delegate(IssueTime, StartTime, EndTime, sa-abc,
    X, canDo(Z, accessDB(db5),true),
            role(X,designEngineer),false)
   ```

4. Marty requires some information from database, db5, at XYZ. He sends a request to SA-ABC along with his certificate.

   ```
   request(marty,accessDB(db5))
   ```

5. SA-ABC knows that the request is from Marty because of his certificate. It then checks the rules to see if Marty as a Design Engineer has access to db5. As this is true, SA-ABC sends a request to SA-XYZ with its certificate. This message says that Marty requires some information from db5 and includes Marty's certificate.
6. SA-XYZ verifies both the certificates and checks it policy to see if SA-ABC has the right to delegate the right to access. As SA-ABC does have the right to delegate, SA-XYZ approves the access and sends the request to the agent controlling access to the database.
7. If Harry, a programmer at ABC, tries to access the database, db5, his request will fail because the SA-ABC has only given design engineers the right.

Request for Authorization This request is very similar to the Request for Action differing in the request for permission used. Also the security agent sends the requester a certificate containing the authorization. As long as the certificate is valid, the agent can access the right without the security agent having to go through the whole reasoning process again. After a request for authorization, the agent can disconnect and then perform a request for action.

The example below describes the authorization process in detail. In this case, Harry, a programmer at ABC, requires some information from database, db5, at XYZ his client.

1. SA-XYZ accesses the company policy for XYZ and the global shared policy.
2. SA-ABC accesses the company policy for ABC and the global shared policy.
3. SA-XYZ sends message to SA-ABC saying that SA-ABC has the right to delegate access to db5, which is a database in XYZ, to all employees.

   ```
   tell(sa-xyz,sa-abc,
        idelegate(StartTime, EndTime, sa-xyz,
          sa-abc, canDo(X,accessDB(db5),
          employee(X,abc)), true,true))
   ```

   SA-ABC asserts the proposition into its own knowledge base.

   ```
   delegate(IssueTime, StartTime, EndTime, sa-xyz,
            sa-abc, canDo(X,accessDB(db5),
            employee(X,abc)),true,true)
   ```

   Then, SA-ABC decides to give all Design Engineers the right to access the database and the right to delegate this right further.

   ```
   tell(sa-abc,sa-abc,
       idelegate(StartTime, EndTime, sa-abc, X,
         canDo(Z, accessDB(db5),true),
         role(X,designEngineer),true))
   ```

   This causes a delegate statement to be inserted

   ```
   delegate(IssueTime, StartTime, EndTime, sa-abc,
            X, canDo(Z, accessDB(db5),true),
            role(X,designEngineer),true)
   ```

4. Harry, a programmer, needs to use a database, db5, from XYZ. He requests his supervisor, Marty, for permission to access db5.

   ```
   requestPermission(harry,marty,accessDB(db5))
   ```

5. Marty is a design engineer and he gives all programmers the right to access db5.

   ```
   tell(marty,sa-abc,
       idelegate(StartTime, EndTime, marty,X,
         canDo(X,accessDB(db5),true),
         role(X,programmer),false))
   ```

   SA-ABC asserts the following clause :

   ```
   delegate(IssueTime,StartTime,EndTime, marty,X,
            canDo(X,accessDB(db5),true),
            role(X,programmer),false)
   ```

6. Now Harry sends a request to SA-ABC along with his certificate

   ```
   request(harry,accessDB(db5))
   ```

7. SA-ABC knows that the request is from Harry because of his certificate. It then checks the rules to see if Harry, as a programmer, is allowed to access db5. Marty has given all

programmers access to db5, so Harry has the right. SA-ABC sends a request to SA-XYZ with its certificate. This message says that Harry requires some information from db5 and includes its own and Harry's certificate.

8. SA-XYZ verifies that the message is coming from SA-ABC and if requires double checks Harry's id. It can also log the request. But it trusts SA-ABC and the request is approved and sent to the access agent controlling the database.

Implementation Details

All the agents are Java Servlets that communicate using HTTP. The knowledge base is Prolog (Swedish Institute 2001b) and the reasoning is carried by rules written in Prolog. The policy is also encoded in Prolog. The security agents have a Jasper (Swedish Institute 2001a) interface to Prolog.

Future Work

We are trying to model obligations and actions to be carried out if an agent fails to fulfill its obligations. We are planning to associate a level of trust with each agent, and modify that based on the fulfillment of obligations.

For our implementation, we had concentrated on Prolog. Now we are upgrading to XML (XML 2000) for describing rights, delegations and authorizations, and XML signatures (XML-Signature 2000) instead of X.509 certificates. We are also working on other issues related to Distributed Trust Management. If an agent is able to access certain public policies of the agent that is in charge of authorization, then it will be in a better position to fulfill those requirements. This leads to the problem of dividing the policy into private and public sections and to the problem of making the public policy available. We are still deciding whether the policy should be made downloadable through HTTP or sent to any agents that request it. Another possible improvement would be for a security agent to return a list of rules that it used to come to the decision, in case the authorization process fails. This allows the requester to figure out where its credentials failed and correct the faults.

Conclusion

The central idea of the paper is to use a system of rights and delegations along with certificates to facilitate trust management. The requester can access a foreign resource by providing its identity information to the agent controlling the resource along with any delegations it may have. The agent controlling the resource uses its policies to verify the identity and delegations of the requester, granting it permission only if everything is valid. We were able to evaluate our infrastructure and interaction protocols by implementing a multi-agent scenario, EECOMS. We also believe that this infrastructure will be very helpful in the home/office automation scenario where the mobile user has to be authorized before she/he can use any service.

References

Blaze, M.; Feigenbaum, J.; Keromytis, A.; and Ioannidis, J. 1998. The keynote trust-management system.

Blaze, M.; Feigenbaum, J.; and Keromytis, A. D. 1999. The role of trust management in distributed systems security. In *Secure Internet Programming*, 185–210.

Bluetoothwebsite. 2001. The official bluetooth website.

Chen, H., J. A. F. T., and Chakraborty, D. 2001. Dynamic service discovery for mobile computing: Intelligent agents meet jini in the aether.

Ellison, C. M.; Frantz, B.; Lampson, B.; Rivest, R.; Thomas, B. M.; and Ylonen, T. 1998. SPKI certificate theory. Internet Draft.

FIPA. 1998. Fipa 98 specification.

Grosof, B., and Labrou, Y. 1999. An approach to using xml and a rule-based content language with an agent communication language.

Herzberg; Mass; Mihaeli; Naor; and Ravid. 2000. Access control meets public key infrastructure, or: Assigning roles to strangers. In *RSP: 21th IEEE Computer Society Symposium on Research in Security and Privacy*.

Ingersoll Rand, Q. 2000. Ciimplex consortium, consortium for integrated intelligent manufacturing planning and execution.

J.K.Jan, C.C.Chang, S. 1991. A dynamic key-lock-pair access control scheme. *Computer and Security* 10.

Johnston, W., and Larsen, C. 1996. A use-condition centered approach to authenticated global capabilities: Security architectures for large-scale distributed collaboratory environments. http://www-itg.1bl.gov/Security/Arch/publications.html.

Lalana Kagal, Vlad Korolev, H. C. A. J. T. F. 2001. Centaurus : A framework for indoor mobile services. *International Conference on Distributed Computing Systems, April 2001*.

Li; Feigenbaum; and Grosof. 1999. A logic-based knowledge representation for authorization with delegation. In *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop*. IEEE Computer Society Press.

Lupu, E., and Sloman, M. 1997. A policy based role object model.

Lupu, E. C.; Marriott, D. A.; Sloman, M. S.; and Yialelis, N. 1995. A policy based role framework for access control.

M.Blaze, J.Feigenbaum, J. 1996. Decentralized trust management. *IEEE Proceedings of the 17th Symposium*.

M.S.Hwang, W.G.Tzeng, W. 1994. A two-key-lock-pair access control method using prime factorization and timestamp. *IEICE Transactions Inf. and Syst* E77-D No.9.

Sandhu, R. S.; Coyne, E. J.; Feinstein, H. L.; and harles E. Youman, C. 1996. Role-based access control models. *IEEE Computer* 20(2):38–47.

Swedish Institute, S. I. o. C. S. 2001a. Jasper.

Swedish Institute, S. I. o. C. S. 2001b. Sicstus prolog.

XML-Signature. 2000. Xml-signature syntax and processing, w3c candidate recommendation 31 october 2000.

XML. 2000. Extensible markup language (xml) 1.0 (second edition) w3c recommendation 6 october 2000.