A Representation Model of Trust Relationships with Delegation Extensions

Isaac Agudo, Javier Lopez, and Jose A. Montenegro

Computer Science Department, E.T.S. Ingenieria Informatica
University of Malaga, Spain
{isaac, jlm, monte@lcc.uma.es}

Abstract. Logic languages establish a formal framework to solve authorization and delegation conflicts. However, we consider that a visual representation is necessary since graphs are more expressive and understandable than logic languages. In this paper, and after overviewing previous works using logic languages, we present a proposal for graph representation of authorization and delegation statements. Our proposal is based on Varadharajan et al. solution, though improve several elements of that work. We also discuss about the possible implementation of our proposal using attribute certificates.

Keywords: Attribute Certificate, Authorization, Delegation, Graph representation, Privilege Management Infrastructure (PMI)

1 Introduction

Traditional authorization schemes are typically based on the fact that the stakeholder has privileges to access the computer resource. That type of systems require that all users of a shared resource are locally registered. However, that requirement conforms a scenario where the authorization service does not scale well.

There are recent schemes where the authorization relies on elements that are external to the functional elements of the computer system. In these schemes, it is necessary to establish a trusted relation among the functional elements and the elements responsible for the authorization. On the other hand, they facilitate the situation in which several computer systems can share the same authorization method. Hence, the global information system results more scalable. We refer to this as *distributed authorization*: the authorization system publishes the authorization elements and the functional system use them to evaluate users' operations.

In many circumstances, delegation is a basic and necessary component to implement distributed authorization. The problem is that actual systems do not represent delegation concepts in a correct way. Several approaches have been proposed in order to manage delegation in distributed environments. The classic solution for knowledge reasoning is the use of logic, although there are other proposals that use a graphical representation.

In this paper we introduce *Weighted Trust Graphs* (WTG) a graphical representation for authorization and delegation. As it can be deduced from its name, it is based on graphs, and provides a more flexible solution, in several ways, than other proposals, like those by Varadharajan and Ruan [1, 2]. One advantage of our solution is that WTG is a generalization of the other proposals. Additionally, WTG allows to define more complex policies. Even if in other solutions a delegation statement is usually issued together with an authorization statement, our solution can use both of them separately, allowing us to introduce the notion of negative delegation. We define negative and positive delegation statements as trust on negative and positive authorization, respectively. Moreover, we implement delegation as transitive trust.

The rest of the paper is structured as follows. Section 2 presents two variants to represent authorization and delegation statements: logic languages and visual representation using graphs. Section 3 describes our proposal based on graph representation, including the formalization of the system. In section 4, we discuss about the possible implementation of our proposal using attribute certificates. Finally, section 5 concludes the paper.

2 Representing Authorization and Delegation Statements

2.1 Logic-based Schemes

This subsection introduces two formal proposals for delegation. These works are based on the use of logic languages to represent the authorization and delegation concepts. The Two significant proposals are, *Delegatable Authorization Program* and *RT Framework*.

Delegatable Authorization Program - DAP. Ruan et al proposed in [3] a logic approach to model delegation. Their language, $\mathcal{L}$, is a first order language, with four disjoint *sorts* $(S, <_S)$, $(O, <_O)$, $(A, <_A)$, and $T = \{-, +, *\}$, for *subject*, *object*, *access right* and *authorization* types, respectively.

In the constant set of authorization types, $T = \{-, +, *\}$, $-$ means negative, $+$ means positive, and $*$ means delegatable. A negative authorization specifies the access that must be forbidden, while a positive authorization specifies the access that must be granted. A delegatable authorization specifies the access that must be delegated as well as granted. The partial orders $<_S$,$<_O$,$<_A$ represent inheritance hierarchies of subjects, objects and access rights, respectively.

In DAP, *predicates* consist of a set of ordinary predicates defined by users, and one built-in predicate symbol, *grant*, for delegatable authorization. The later is a 5-term predicate symbol with type $S \times O \times T \times A \times S$, where the first argument is the grantee, the second one is the object, the third is the authorization type, the fourth is the access right and, finally, the fifth argument is the grantor of this authorization. Intuitively, $grant(s, o, t, a, g)$ means s is granted by g the access right a on object o with authorization type t. *grant* is called *authorization predicate*. There are two special predicates named *cangrant* and *delegate*, of type $S \times O \times A$ and $S \times S \times O \times A$, respectively, that are used to model delegation. $cangrant(s, o, a)$ means subject s has the right to grant access a on object o to other subjects, while $delegate(g, s, o, a)$ means subject g has granted to subject s access a on object o with access type $*$.

A DAP consists of a finite set of rules of the form:

$$b_0 \leftarrow b_1, \ldots, b_k, not\, b_{k+1}, \ldots, not\, b_{k+m},\ m \geq 0$$

In [6], Ruan et al. extend their model with temporal capabilities, adding a new temporal parameter to predicates.

Example. Sorting of the language is defined as, $S = \{\#, s_1, s_2 : s_1 <_S s_2\}$, $O = \{o_1, o_2; o_1 <_O o_2\}$, $A = \{write, read; write <_A read\}$, and the rules of the DAP as:

```
r1 : dba($s_1$) ←
r2 : ¬dba($s_2$) ←
r3 : ¬secret($o_1$) ←
r4 : secret($o_2$) ←
r5 : grant($s_1$,$o_2$,*,write,#)←
r6 : grant($s_2$,$o_2$,-,write,$s_1$) ←
r7 : grant(_s,_o,-,write,#) ← secret(_o),not dba(_s)
```

RT Framework. Li et al. proposed in [5] logic programming as a way to model authorization and delegation relations. They use *Roles* for this purpose and they define a full general framework, RT for *Role Based Trust Management*. It comprise five different solutions, each of them with different characteristics. Roles can be interpreted as privileges or attributes, and are similar to what Ruan et al. call *access rights*. As in the previous logic-based proposal, the RT Framework defines a partial order in roles, establishing how rights can be inherited. Partial orders are used to represent other concepts too. Let u, p, r denote users, rights and roles, respectively; then:

- $r_1 \geq r_2$, is read as r_1 *dominates* r_2, and means that r_1 has all the rights r_2 has. It can also be read as r_2 contains r_1.
- $u \geq r$ assigns role r to user u.
- $r \geq p$ assigns right p to role r.

RT defines several types of credentials, an analogous concept to *DAP*. The basic credentials are:

1. $A.R \leftarrow D$: This credential means that A defines D to be a member of A's role R. In the attribute-based view, this credential can be read as D *has the attribute* $A.R$, or equivalently, A *says that* D *has the attribute* R.

2. $A.R \leftarrow B.R_1$: This credential means that A defines its role R to include all members of B's role $R1$. In the attribute-based view, this credential can be read as *if B says that an entity has the attribute R_1, then A says that it has the attribute R.*
3. $A.R \leftarrow A.R_1.R_2$: The expression on the right is called a *linked role*. It means that $A.R$ contains $B.R_2$ for all B in $A.R_1$. The attribute-based reading of this credential is: *if A says that an entity B has the attribute R_1, and B says that an entity D has the attribute R_2, then A says that D has the attribute R.*
4. $A.R \leftarrow B_1.R_1 \cap B_2.R_2 \cap \ldots \cap B_n.R_n$: This credential means that if an entity is a member of $B_1.R_1$, $B_2.R_2$, ... and $B_k.R_k$, then it is also a member of $A.R$. The attribute-based reading of this credential is *A believes that anyone who has all the attributes $B_1.R_1$, ..., $B_k.R_k$ also has the attribute R.*

```
EPub.disct ← EPub.preferred∩EPub.student
EPub.preferred ← EOrg.preferred
EOrg.prederred ← IEEE.member
EPub.student ← EPub.university.stuID
EPub.university ← ABU.accredited
ABU.accredited ← StateU
StateU.stuID ← Alice
IEEE.member ← Alice
```

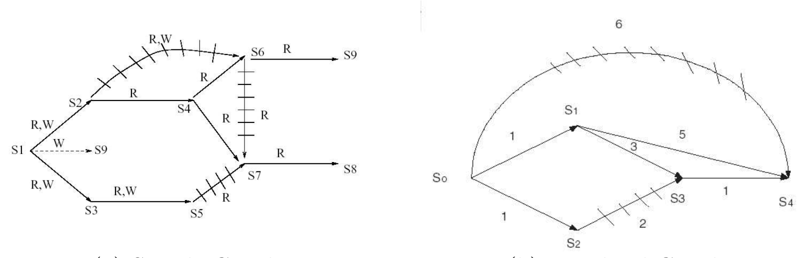
2.2 Graphs-based Schemes

Logic programming offers a powerful mechanism to represent authorization and access control decisions. Authorizations are represented as predicates and decisions are based on formulae verification. There are many logical solutions for formulae verification and it is easy to implement such a system in a standard way. A disadvantage of logical programming is that it is not well understandable and has an obscure transcription. The previous solutions are clear examples.

On the other hand, there are graphical solutions that are thought to be less powerful but more expressive and more understandable. A graphical solution may be based on the use of directed graphs to model authorization and delegation process. Basically, this maps each predicate to a directed arc in a graph. Arcs goes from the issuer of the authorization or delegation statement to the subject who is authorized or granted privileges. There are as many different arcs as different authorization/delegation statements to consider. As we model authorization and delegation, the graph we get is a tree and the root of the tree is (usually) the owner of the resource which we are reasoning about. With such a tree it is possible to study the relations between entities in the system in a graphical way.

Varadharajan and Ruan have proposed two solutions to represent authorization and delegation using directed graphs. In [1] they present a first approach to the problem. This approach considers three types of authorizations: negative authorization, positive authorization and delegatable authorization. As shown in figure 1, a cross line represents a negative authorization, a dashed line represents a positive authorization, and a simple line represents a delegatable one.

In [2] the same authors proposed a new approach, *weighted graphs*. In that proposal, each authorization is associated with a weight given by the grantor, representing the degrees of certainties about the authorization grants. The weight is a non-negative number, and a smaller number represents a higher certainty. When considering both negative and positive authorizations, we get conflicts if the same subject is issued a negative and a positive authorization. In this case, we need to define a conflict resolution method that allows us to decide which of them has to be considered.



(a) Simple Graph (b) Weighted Graph

Fig. 1. The two graph-based models proposed by Varadharajan et al.

These authors follow the idea of predecessor-take-precedence. However, there are still some conflicts that they do not solve. For instance, Figure 2 shows an incomparable conflict in the weighted graphs approach. We believe that ACD should override ABD because in the first link of the path A prefers C instead of B. This follows the predecessor-take-precedence philosophy as previous decisions take precedence over later ones. As shown later, our solution resolves this conflict.

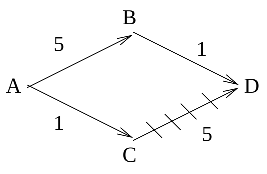


Fig. 2. Incomparable conflict

3 Weighted Trust Graph

Weighted Trust Graphs (WTG) aims to generalize the proposal presented in [2]. However, our definition is more flexible. In fact, we support this proposal as a particular case of our framework.

As we consider negative and positive authorizations, conflicts between them may arise. We assign to each authorization a weight that, together with the security level policy, allow us to avoid many conflicts. In case the weights are the same, we follow a predecessor-take-precedence principle with some refinements; that is, a new conflict resolution method, that we call *strict-predecessor-take-precedence.*

This principle can also be used as a stand alone policy, where the owner of the resource establishes a hierarchy of subjects by assigning appropriated weights to their delegations, and any of the further delegations made for these subjects has to preserve this hierarchy. For instance, if A gets from S the higher priority in the hierarchy, all A's delegation or authorization statements take preference over the others ones.

In this paper we propose a security policy to avoid conflicts. We call it *Mean Policy*, where we use the *strict-predecessor-take-precedence* principle to solve conflicts.

Credentials are represented using arcs in a graph. Thus, both terms are used equally. We consider a credential as a 4-tuple:

$$(Issuer, Subject, Type, Right)$$

where (i) $Issuer$ is the issuer of the authorization or delegation statement; (ii) $Subject$ is whom this statement refers to; (iii) $Type$, as we will see later, is the type of the statement in a general way; and, (iv) $Right$ is the right together with the resource we are reasoning about. We can represent $Right$ as a 2-tuple consisting on the resource and the type of access, $Right = (Resource, Access)$. As with the $Right$ of a credential, we can also express the $Type$ as a 3-tuple[1] composed of the following parameters:

- $Weight$, which represents the level of trust in this authorization.
- $Delegatable$, which represents if the statement is delegatable or not.
- $Sign$, which represents the sign of the statement (negative or positive).

Then, we can define a credential as follows.

Definition 1. *A* ***credential*** *is a 4-tuple of the form* ($Issuer$, $Subject$, $Type$, $Right$) *where* $Issuer \in \mathcal{S}$, $Subject \in \mathcal{S}$, $Type = (w, d, s) \in \mathcal{D} \times \{0,1\} \times \{0,1\}$ *and* $Right = (o, a) \in \mathcal{O} \times \mathcal{A}$.

- $\mathcal{S}$ *is the set of subjects in the system;*
- $\mathcal{D}$ *is the domain where we evaluate the credential. In general, it could be any real number, but for our framework we restrict it* [2] *to* $\mathcal{D} = [0, 1]$. *We consider*

[1] If we consider validity intervals, it should be part of the credential $Type$ and we should use a 4-tuple.

[2] Varadharajan et al. proposal could be derived from our framework, using non-negative numbers instead of $[0, 1]$.

it as the level of trust that the issuer has on this credential: '1' stands for fully trustable credential, while '0' stands for non trustable credentials.
- *$\mathcal{O}$ is the set of objects;*
- *$\mathcal{A}$ is the set of access types.*

We denote the set of all credentials by $\mathcal{G}$. Given a credential, we can refer to its *Type* components by the functions described next.

Definition 2. *Let m be a credential, then:*

- ***Weight**, $|m|$ defines the weight of the arc; according to Definition 1, $|m| \in [0, 1]$. When $m = 0$ we consider this credential as non existing.*
- ***Delegatable**, $d(m)$ is '0' for authorizations, and '1' for delegation statements.*
- ***Sign**, $s(m)$ is the sign of the credential ('1' if positive, '0' if negative).*

Based on the proposal by Varadharajan et al., we define a graphical representation for the four types of credentials that we can obtain for the parameters d and s. Parameter w (the weight) is placed over the arcs in the graph. The different arc types that we support are represented in figure 3:

- Arc a) represents a positive delegation statement; i.e. $d(m) = 1$ and $s(m) = 1$. It means that the issuer trusts the subject about his/her positive authorizations or delegations. For simplicity, we suppose that this credential can be interpreted as a b or c credential.
- Arc b) represents a positive authorization statement; i.e. $d(m) = 0$ and $s(m) = 1$. It means that the issuer authorizes the subject to access the resource.
- Arc c) represents a negative delegation statement; i.e. $d(m) = 1$ and $s(m) = 0$. It means that the issuer trusts the subject about his/her negative authorizations or delegations.
- Arc d) represents a negative authorization statement; i.e. $d(m) = 0$ and $s(m) = 0$. It means that the issuer denies access to the subject over the resource.

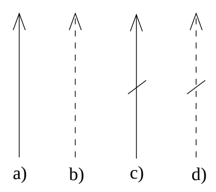


Fig. 3. Different types of arcs

We allow negative delegation statements which stands for trust about negations. In the example of figure 4, a bank could issue a negative delegation statement over granting credits to a company that maintains a customer blacklist, and at the same time could issue a (positive) delegation statement to one of its local offices. Even if the local office agrees on granting a credit to a specific citizen, they have to take into account the negative authorization issued by the black list company.

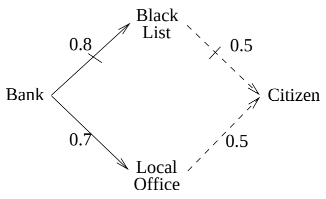


Fig. 4. Negative delegation

When taking decisions about granting access to a certain resource, we have to analyze the chains or paths of credentials from the owner of the resource to the subject. Next, we define paths of credentials.

Definition 3. *A* ***path*** $\mathcal{C}$ *is a sequence of consecutive credentials,* $m_1 m_2 \ldots m_n$*. Every* m_i *is a credential and by 'consecutive' we mean that the issuer of* m_{i+1} *must be the subject of* m_i *for all* $i \in \{1, \ldots, n-1\}$*. The* ***length*** *of the path is defined as the number of credentials.*

If there is only one path between the nodes, we can also note a path by its nodes instead of its arcs. Once we define a path, there could be conflictive paths between the same subjects, so we have to be able to compare them and decide which of them is valid and which not.

The key for conflict resolution is to define a metric over paths of credentials, which allow us to measure the priority of each authorization or, at least, to compare them. Such a metric should follow the predecessor-takes-precedence method. Therefore, a path should become less important as it grows (or, what is to say, as new arcs are added). We will use the term *metric*, even if it is not a metric in the mathematical sense.

Definition 4. *A* ***metric*** *is a non-negative function* $|\mathcal{C}| = f(|m_1||m_2| \ldots |m_n|)$ *where* $n = length(C)$*. In case there are different metrics, we denote it as* $|\cdot|_i$*, where the subindex refers to the particular metrics.*

The main property a metric has to fulfill in order to be considered as a metric for paths is *monotony.* This means that the weight or level of trust of the path decreases when going down in the path. As we want that larger paths are less important, we are interested in decreasing metrics, i.e., if we add one more credential to an existing path, the weight of the newer path can not be greater than the weight of the older path:

$$|m_1 m_2 \ldots m_i| \geq |m_1 m_2 \ldots m_i m_{i+1}| \tag{1}$$

Some examples of monotone metrics are,

- $|\mathcal{C}|_. = |m_1||m_2|\cdots|m_n|$
- $|\mathcal{C}|_{min} = min(|m_1|, |m_2|, \ldots, |m_n|)$
- $|\mathcal{C}|_+ = |m_1| + |m_2| + \ldots + |m_n|$
- $|\mathcal{C}|_{max} = max(|m_1|, |m_2|, \ldots, |m_n|)$

The first two are decreasing metrics[3], and the last two are increasing metrics. With the help of these metrics we can define an order in the paths as follows:

$$\mathcal{C} > \mathcal{C}' \text{ if } |\mathcal{C}| > |\mathcal{C}'| \tag{2}$$

Although there is a variety of orders that can be defined in this way, others can not. One example is the lexicographic or dictionary order.

Definition 5. *Given two paths $\mathcal{C}$ and $\mathcal{C}'$, let n be the minimum length of the two, $n := min\{lenght(\mathcal{C}), lenght(\mathcal{C}')\}$, we say that $\mathcal{C} >_L \mathcal{C}'$ if*

$$\mathcal{C} >_L \mathcal{C}' \, if \begin{cases} (|m_1| > |m'_1|) \\ or \\ (|m_1| = |m'_1|) \wedge |m_2| > |m'_2| \\ or \\ \ldots \\ or \\ (|m_1| = |m'_1|) \wedge \ldots \wedge (|m_{n-1}| = |m'_{n-1}|) \wedge |m_n| > |m'_n| \\ or \\ |m_i| = |m'_i| \forall i \in \{1, \ldots, n\} \wedge n = lenght(\mathcal{C}) \end{cases}$$

When inspecting paths from one node to another, not all the possible paths need to be considered. It is necessary to define which are valid paths and which are not. If A trusts $B's$ negative authorizations, then paths containing $B's$ positive authorizations over $A's$ resources have to be discarded. If A authorizes B and B authorizes C, C is not authorized, as A does not trust B on positive authorizations. He just authorizes him, thus this path is also discarded. Next we define *valid paths.*

Definition 6. *Given a path, $\mathcal{C} = m_1 m_2 \ldots m_n$, we say that $\mathcal{C}$ is valid or consistent if any of the following conditions holds:*

- *$length(\mathcal{C}) = 1$*
- *$d(m_i) = 1$ for all $i \in \{1, .., n-1\}$ and $s(m_i) = -1$ for all $i \in \{1, .., n\}$*
- *$(d(m_i) = 1) \wedge (s(m_i) = 1) \forall i \in \{1, .., n-1\}$*

We define the function v to determine whether a path is valid or not,

$$v(\mathcal{C}) := \begin{cases} 1 \text{ } if \, \mathcal{C} \text{ } is \text{ } consistent \\ 0 \text{ } in \text{ } other \text{ } case \end{cases}$$

[3] $|\cdot|_.$ is decreasing as we define $0 \leq |m|_. \leq 1$ for all credentials in the system (see Definition 2)

We have to keep in mind that, when taking access decisions, the last link in the path the user presents consists of an authorization and, therefore, provides the sign (positive or negative) for the path. We will consider that a path is positive/negative if the last link is positive/negative, respectively. We define, based in a given metric, a *pseudo-metric*, that takes into account all the details we have defined before. We denote it with double lines.

Definition 7. *Given a metric $|\cdot|$, a sign function $s(\cdot)$, a validity function $v(\cdot)$ and a path $\mathcal{C} = m_1 m_2 \dots m_n$, we define the associated* ***pseudo-metric*** *as*

$$\|\mathcal{C}\| = |\mathcal{C}| s(m_n) v(\mathcal{C})$$

As $|\mathcal{C}| \in [0,1]$, $\|\mathcal{C}\| \in [-1,1]$ then, given a number $x \in [-1,1]$, we define $\|x\|^{-1}$ as the set of all paths with pseudo-weight equal to x.

$$\|\mathcal{C}\|^{-1} := \{\mathcal{C} \in \mathcal{G} : \|\mathcal{C}\| = x\} \tag{3}$$

We choose for our proposal the metric, $|\mathcal{C}| = |\mathcal{C}|$. and we omit the dot in the following.

Definition 8. *Given a set of paths $\mathcal{S}$, and the pseudo-metric $\|\mathcal{C}\|$, we define the* ***highest*** *and the* ***lowest*** *weight of the set $\mathcal{S}$ as,*

$$\mathcal{H}(\mathcal{S}) := max\{\|\mathcal{C}\| : \mathcal{C} \in \mathcal{S}\}$$

and

$$\mathcal{L}(\mathcal{S}) := min\{\|\mathcal{C}\| : \mathcal{C} \in \mathcal{S}\}$$

respectively. If $S = \emptyset$ then they are defined to be equal to '0'.

In particular, we can define the *higher/lower* weight between two nodes, as shown next.

Definition 9. *Let S_{AB} be the set of all valid paths ($v(\mathcal{C}) \neq 0$) from A to B, then $\mathcal{H}_{AB} := \mathcal{H}(S_{AB})$ and $\mathcal{L}_{AB} := \mathcal{L}(S_{AB})$*

We can also define the average weight, compensating negative and positive authorization.

Definition 10. *Let S_B be the set of all credentials issued over B, and let X_m be the issuer of the credential m. Then, the average weight between principals A and B is defined as,*
$\mathcal{M}_{AB} := average\{|m| s(m) \mathcal{M}_{AX_m} : m \in S_B \wedge \mathcal{M}_{AX_m} > 0\}$
The initial case is $\mathcal{M}_{AA} := 1$.

Definition 10, is a recursive definition that can be seen as a graph exploration using a *branch and bound* algorithm where the strict lower bound for $\mathcal{M}_{AX}$ is set to zero. If we calculate $\mathcal{M}_{AB}$ from A to B, in the first step we inspect the principals conected from A with a single arc. Then we mark the negatives as "non useful", as they can not further delegate, so can not be part of any delegation path. With the rest we repeat the argument until we reach B. At the end, we have marked all the non useful nodes. When reasoning about two principals A and B, the non useful nodes are omitted and we get an effective graph containing only the useful nodes.The resulting graph is easier to inspect both visually and aritmetically.

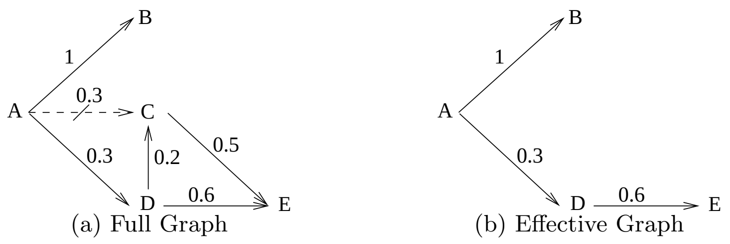


Fig. 5. An example of a Delegation Graph and the Effective Graph

In the following, we will explain how to calculate the previously defined functions with an example. We are going to calculate $\mathcal{M}_{AX}$, for all relevant nodes X in the full graph of Figure 5,

$$\mathcal{M}_{AB} = 1 \times 1 \times \mathcal{M}_{AA} = 1 \tag{4}$$

$$\mathcal{M}_{AD} = 0.3 \times 1 \times \mathcal{M}_{AA} = 0.3 \tag{5}$$

$$\begin{aligned}\mathcal{M}_{AC} &= \frac{0.3 \times (-1) \times \mathcal{M}_{AA} + 0.2 \times 1 \times \mathcal{M}_{AD}}{2} \\ &= \frac{-0.3 + 0.2 \times 0.3}{2} \\ &= -0.12\end{aligned} \tag{6}$$

$$\mathcal{M}_{AE} = 0.6 \times (1) \times \mathcal{M}_{AD} = 0.6 \times 0.3 = 0.18 \tag{7}$$

What we actually do is the arithmetic average of all valid paths with all its nodes, X_i, having a positive $\mathcal{M}_{AX_i}$. Note that we have not considered the path ACE in the calculus of $\mathcal{M}_{AE}$ as $\mathcal{M}_{AC} < 0$ (see Definition 10 for more details). With the same example it is easy to calculate $\mathcal{L}_{AX}$ and $\mathcal{H}_{AX}$.

With these three elements we can define several types of policies for authorization. At this moment, we are working on a classification of the policies. One of the simplest policies is the **Mean Policy**. Easy speaking, A grant access to the resource to agent X if $\mathcal{M}_{AX} > 0$. Although this is an easy formula, there are cases in which $\mathcal{M}_{AX} = 0$, thus this formula does not grant authorization but, according to predecessor-takes-precedence method, we should grant it or implicitly deny it.

When $\mathcal{M}_{AX} = 0$, it could happen that there is no path from A to X or that they are both positive and negative. As the average is zero, we can not decide whether to grant authorization or not. Other proposals keep this situation as unsolvable. However, it is important to note that we try to solve it using the lexicographic order to decide about granting authorization.

Given a set of paths, we can also define another subset, including all the maximal elements according to the lexicographic order. We note it as $max_L(AB)$.

Definition 11. *Given a set of path, S, we define the* ***maximal lexicographic subset*** *of S as $max_L(S) := \{\mathcal{C} \in S : \text{ there are no } \mathcal{C}' \in S \text{ with } \mathcal{C}' >_L \mathcal{C}\}$.*

If the set S is the set of all paths from A to B, then we denote it as $max_L(AB)$

Using this set we can solve several cases in which $\mathcal{M}_{AX} = 0$. We allow access if any of the paths with the highest weight is greater (using the dictionary order) than all the paths with the lowest weight. In other case, we say the conflict in undecidable.

Now we can formally define this policy.

Definition 12. *According to the* ***Mean Policy, A*** *grants access to their resources to agent* ***X*** *if $\mathcal{M}_{AX} > 0$ or $\mathcal{M}_{AX} = 0$ and*

$$\exists\, \mathcal{C} \in \|\mathcal{H}_{AB}\|^{-1} : \; \mathcal{C} >_L \mathcal{C}' \;\forall\, \mathcal{C}' \in \|\mathcal{L}_{AB}\|^{-1}$$

Using this policy we can solve the problems mentioned when explaining Figure 2.

As mentioned in the introduction we also define a stronger principle than *predecessor-take-precedence.* We call it *strict-predecessor-take-precedence* and we implement it using the dictionary order. We order all path from A to X and choose the maximal ones; these are the preferred paths. If all of them are positive, then we grant authorization, but if there are positive and negative we can not say anything[4]. If there are only negative authorizations we deny access.

Definition 13. *According to* ***strict-predecessor-take-precedence***, ***A*** *grants access to their resources to agent* ***X*** *if $max_L(AX)$ contains only positive authorizations.*

4 Implementation of our proposal using Attribute Certificates

The last X.509 ITU-T Recommendation [7] introduces the concept of *Privilege Management Infrastructure* (PMI) as the framework for the extended use of *attribute certificates.* The Recommendation establishes four PMI models: (i) *General*, (ii) *Control*, (iii) *Roles* and (iv) *Delegation.* The first one can be considered as an abstract model, while the other ones can be considered as the models for implementation.

[4] In fact, we can solve it combining the two policies presented in the paper or using other policies we are working on.

The implementation of the Control and Roles models are feasible tasks, though not free of complexity. However, the case of the Delegation model is substantially different because of the intrinsic difficult problems of the delegation concept. In this section, we discuss about the implementation of the Delegation model using our WTG solution in combination with attribute certificates.

The PMI area inherits many notions from the *Public Key Infrastructure* (PKI) area. In this sense, an *Attribute Authority* (AA) is the authority that assigns privileges (through attribute certificates) to users, and the *Source of Authorization* (SOA) is the root authority in the delegation chain. A typical PMI will contain a SOA, a number of AAs and a multiplicity of final users. As regarding our scheme, we will represent the previous elements as the nodes of the graph. The SOA will be the first node that outflows initial arcs. AAs will be the intermediary nodes while the final users will be the leaf nodes (that is, the nodes that do not outflow arcs but inflow authorization arcs only).

The ASN.1 [4] description of the structure of an attribute certificate is the following:

```
AttributeCertificate ::= SIGNED {AttributeCertificateInfo}
AttributeCertificateInfo ::= SEQUENCE
 {
    version                    AttCertVersion, --version is v2
    holder                     Holder,
    issuer                     AttCertIssuer,
    signature                  AlgorithmIdentifier,
    serialNumber               CertificateSerialNumber,
    attrCertValidityPeriod     AttCertValidityPeriod,
    attributes                 SEQUENCE OF Attribute,
    issuerUniqueID             UniqueIdentifier OPTIONAL,
    extensions                 Extensions  OPTIONAL
}


Extensions ::= SEQUENCE OF Extension
Extension ::= SEQUENCE {
    extnId      EXTENSION.&id ({ExtensionSet}),
    critical        BOOLEAN DEFAULT FALSE,
    extnValue       OCTET STRING
}


ExtensionSet EXTENSION      ::= {  ... }
EXTENSION ::= CLASS {
    &id     OBJECT IDENTIFIER UNIQUE,
    &ExtnType }
    WITH SYNTAX {
    SYNTAX          &ExtnType
    IDENTIFIED BY       &id
    }
```

One of the fields is the *extensions* field. This is precisely the field than becomes essential for the practical implementation of our proposal. This field allows us to include additional information into the attribute certificate. The X.509 standard provides the following predefined extension categories:

Basic privilege management: Certificate extensions to convey information relevant to the assertion of a privilege.

Privilege revocation: Certificate extensions to convey information regarding location of revocation status information.

Source of Authority: These certificate extensions relate to the trusted source of privilege assignment by a verifier for a given resource.

Roles: Certificate extensions convey information regarding location of related role specification certificates.

Delegation: These certificate extensions allow constraints to be set on subsequent delegation of assigned privileges.

We focus on the Delegation extension category, that defines different extension fields. Among them, the Recommendation includes:

Authority attribute identifier: In privilege delegation, an AA that delegates privileges, shall itself have at least the same privilege and the authority to delegate that privilege. An AA that is delegating privilege to another AA or to an end-entity may place this extension in the AA or end-entity certificate that it issues. The extension is a back pointer to the certificate in which the issuer of the certificate containing the extension was assigned its corresponding privilege. The extension can be used by a privilege verifier to ensure that the issuing AA had sufficient privilege to be able to delegate to the holder of the certificate containing this extension.

That extension is close to our goals. However, it does not define the weight associated to the arc between the issuer and the holder of the certificate. Therefore, we define our own extension, in ASN.1, based on the *Authority attribute identifier* one.

This new extension determines a sequence between the SOA and the holder. Each sequence includes other sequence, *ArcsId*, where to include the information of the arcs in the graph, weight of the arc, origin node, and boolean information about statements, delegation and sign. The destination node must coincide with the serial number of the attribute certificate.

```
WeightPathIdentifier EXTENSION  ::=
 {
       SYNTAX               WeightPathIdentifierSyntax
       IDENTIFIED BY       { id-ce-WeightPathIdentifier }
 }
WeightPathIdentifierSyntax  ::= SEQUENCE SIZE (1..MAX) OF ArcsId

ArcsId ::= SEQUENCE {
             Origin      IssuerSerial,
             Destination HolderSerial,
             Weight      REAL (0..1),
             Delegable   BIT,
             Sign        BIT
}
```

5 Conclusions and Ongoing Work

Delegation is increasingly becoming a basic issue in distributed authorization. Actual systems do not represent delegation concepts in a correct way, and some approaches have been proposed for the management of delegation. The traditional solution for knowledge reasoning is the use of logic, although there are other proposals that use a graphical representation.

We have presented in this work the Weighted Trust Graphs (WTG) solution, a graphical representation for authorization and delegation. There are other solutions based on graphs. However, WTG provides a more flexible solution because it is a generalization of the other proposals. Additionally, WTG allows to define complex policies. Moreover, our solution can make use of authorization and delegation separately, allowing to manage negative delegations, which stands for trust on negative authorizations.

More complex policies than the one presented in the paper (at the end of section 3) are part of our actual research. The key to define policies is the use of general inequations on variables $\mathcal{H}_{AX}$, $\mathcal{M}_{AX}$ and $\mathcal{L}_{AX}$. These equations can be combined to produce more complex policies, using sequences of equations or systems of equations. Another importan ingredient for constructing policies is the use of the set $max_L(AX)$, or more generally, the use of the dictionay order.

The *Mean Policy* we have presented is the simplest inequation one can consider. Another simple policy, could be

$$\mathcal{L}_{AX} > 0$$

This equation holds only when there are no negative authorizations from A to X and there is at least one positive authorization. We are working on a classification of these policies, and are studying the efficiency of each solution.

In addition to the theoretic definition of the WTG scheme, this work has shown how to perform a practical implementation based on attribute certificates and the PMI framework defined by ITU-T.

6 Acknowledgements

This paper is an outcome of the work performed in three Research Projects where the different co-authors have been involved. We very much thank the support of: (i) the European Commission through the UBISEC Project (IST-506926), (ii) the Japanese National Institute of Information and Communication Technology (NICT) through the International Collaborative Research Project "Secure Privacy Infrastructure" and, (iii) the Spanish Ministry of Science and Technology through the project PRIVILEGE (TIC-2003-8184-C02-01). We also thank the Andalusian Regional Government, that supports Isaac Agudo's PhD work.

References

1. V. Varadharajan, C. Ruan. Resolving conflicts in authorization delegations. In *ACISP*, volume 2384 of *Lecture Notes in Computer Science*. Springer, 2002.

2. V. Varadharajan, C. Ruan. A weighted graph approach to authorization delegation and conflict resolution. In *ACISP*, volume 3108 of *Lecture Notes in Computer Science*. Springer, 2004.
3. Y. Zhang, C. Ruan, V. Varadharajan. Logic-based reasoning on delegatable authorizations. In *Foundations of Intelligent Systems : 13th International Symposium, ISMIS*, 2002.
4. B. Kaliski. *A Layman's Guide to a Subset of ASN.1, BER, and DER*, 1993.
5. N. Li, J. C. Mitchell, W. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.
6. C. Ruan, V. Varadharajan, Y. Zhang. A logic model for temporal authorization delegation with negation. In *6th International Information Security Conference, ISC*, 2003.
7. ITU-T Recommendation X.509. *Information Technology - Open systems interconnection - The Directory: Public-key and attribute certificate frameworks*, 2000.