Implementation Aspects of a Delegation System

Isaac Agudo
Computer Science Department
E.T.S. Ingenieria Informatica
University of Malaga,Spain
isaac@lcc.uma.es

Javier Lopez
Computer Science Department
E.T.S. Ingenieria Informatica
University of Malaga,Spain
jlm@lcc.uma.es

Jose A. Montenegro
Computer Science Department
E.T.S. Ingenieria Informatica
University of Malaga,Spain
monte@lcc.uma.es

ABSTRACT

In this paper we simulate an authorization and delegation system using knowledge based technology. This proposal is part of a visual tool that is intended to be an implementation of the theoretical model weighted trust graph (WTG). A brief description of WTG Model and its associated tool is included in the text. In essence, the model is based on the inclusion of real numbers between zero and one in certificates to represent the trust level between the entities involved in them. This trust level is used to control delegation. Moreover, attributes from different domains may be interrelated, so attribute delegation is also taken into account. The proposed Simulation Engine supports one directional and bidirectional search algorithms.

1. INTRODUCTION

Many authorization systems have been presented in the literature, most of them based on logic formalism. They are mostly based on DATALOG (PROLOG variant) but it is difficult to find real implementations with full implementation details. In most approaches, two logical predicates are used to encode the granting and delegation of attributes. They are usually denoted with *grant* and *delegated* respectively. Certificates or credentials are encoded using predicates also, and delegation is implemented with the appropriate inference rules. In [7, 8] two logic authorization frameworks are defined with different characteristics. There are many differences between them, but the main ones are:

- DAP support negative credentials and hence non-monotonic reasoning whereas RT do not.
- RT is based on local Roles or Attributes, whereas DAP is based on resources and access rights.

We take the best of both, so we try to implement non-monotonic reasoning, i.e. authorization can be retracted, and we make use of local attributes. More over we provide mechanisms to connect attributes from different domains.

MOBIMEDIA 2007, August 27-29, Nafpaktos, Greece
Copyright © 2007 ICST 978-963-06-2670-5
DOI 10.4108/ICST.MOBIMEDIA2007.1901

We have chosen CLIPS to implement the reasoning tool for WTG because it is a widely used tool for knowledge representation and reasoning. Certificates are the facts of the system, together with other useful information, and the inference engine gives us the answer to our authorization requests. Moreover, CLIPS can be called from a C application so connecting CLIPS with the graphical tools developed to define WTG credentials is straightforward. This tool, including its source code, is available for download at http://openpmi.sourceforge.net.

When asking for an authorization request, there are two main approaches. The first approach consists of inferring all authorization predicates and then checking if the desired predicate is within the generated fact set. If the set of possible inferred rules (determined by the number of certificates and rules) is comparable to the normal number of requests per unit of time, this approach, which is called one directional search, is highly recommended. The second approach, or goal oriented approach, makes use of the desired predicate as a compass, so the choosing of inference rules is influenced by the predicate we are looking for. When the number of requests is significantly smaller than the set of potentially inferred facts, is better not to spend time checking non useful predicates and instead to do a goal oriented or bidirectional search.

WTG uses attributes to define authorization. When using attributes, we have to keep in mind that they only have meaning in the domain of definition of the attribute. Attributes are defined by entities, who are in charge of giving them meaning. Attributes are then defined as a pair $(Resp, ID)$. WTG offers the possibility of linking attributes from different domains. In this way, an attribute is said to be subscribed to another attribute when the first attribute inherits all the privileges of the second one. As an example, Alice may define attribute $best\_friend$ to be subscribed to attribute $friend$. In this way $best\_friend$ is defined by Alice as a specification of the attribute $friend$. Moreover, Alice may subscript her attribute $friend$ to the attribute $(Bob, friend)$, so Bob's friend are defined by Alice as her friends.

The outline of the paper is the following. Section two provides a brief description of the theoretical model weighted trust graph (WTG). A visual tool to design and create instances of WTG is described in section three. Section four details the knowledge based system used to simulate the WTG instances designed with the previously described visual tool. Finally, section five covers conclusions and ongoing work.

2. A BRIEF INTRODUCTION TO WEIGHTED TRUST GRAPH (WTG)

Weighted Trust Graphs (WTG) [6] is a formalism that allows authorization and delegation relationships to be modelled. Authorization and delegation credentials have an associated index that is used to define the trust or confidence level between the entities involved regarding the attribute encoded in the credential, e.g. Alice may issue a credential to Bob granting attribute *InternetAccess* with a trust level of 0.5. In a real situation the trust level can be used to change authorization decisions based on context information. There may be cases in which a trust level of 0.5 is enough and cases in which it is not. The trust level can also be used to evaluate the degree of responsibility of the issuer of the credential in the case of a bad use of the attribute.

One of the mayor advantages of WTG is that it allows users to define more complex policies and provides a graphical representation for them. Another difference from previous proposals is that in WTG, delegation statements are defined separately from authorization ones, so a delegation credential does not implicitly give authorization rights.

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

$$(Issuer, Subject, Type, Attribute)$$

where

1. $Issuer$ is the issuer of the authorization or delegation statement,
2. *Subject* is whom this statement refers to,
3. $Type$ is used to include the extra information needed for defining authorization policies in credentials,
4. *Attribute* is the attribute granted or denied to the subject.

The type of a credential consists of a 3-tuple composed of the following parameters:

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

If we want to include time constraints, it should be part of the credential $Type$ and we should use a 4-tuple.

Then, a credential is defined as follows.

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

- $\mathcal{S}$ 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 to $\mathcal{D} = [0,1]$. We consider it as the level of trust that the issuer has on this credential: '1' stands for fully trustable credential, while '0' stands for null or empty credentials.
- $\mathcal{A}$ is the set of Attributes.

Delegation credentials can be chained, resulting in a delegation path. Delegation paths have also an associated weight, computed as the product of the weights of all the certificates in the path. A delegation policy is a criteria used to define whether or not attributes are effectively delegated to a particular user. The mere existence of a delegation certificate is not enough. We represent by $delegation(holder, attrResp, attrId)$ that $holder$ has been delegated attribute $attrId$ by $attrResp$. As mentioned before, attribute $attrId$ by $attrResp$ is different from $attrId$ by $attrResp2$, so attributes only have a local meaning. A simple and efficient delegation policy is used in this paper. It consists of the following criteria, $delegation(holder, resp, attrId)$ holds if the greatest weight of the positive delegation paths from $resp$ to $holder$ is greater than the corresponding of the negative delegation paths.

An authorization path consists of a delegation path plus a consecutive authorization credential. The weight of an authorization path is the product of the weight of the delegation path and the authorization credential. In order to represent that $holder$ is authorized by $attrResp$ to make use of the attribute $attrId$, the predicated $authorization(holder, attrResp, attrId)$ is used. Authorization policies consist of criteria for deciding whether or not a user is authorized to perform a certain operation. WTG defines several authorization policies, we opt here for a pessimistic approach so authorization policies will consist of a lower bound for the minimal authorization path. Authorization policies are defined by attribute managers, who are in charge of restricting usage of the attributes. An authorization policy is represented by $policy(attrResp, attrID, bound)$.

In WTG, users share their own resources with other users and consume other users' services and resources. Our model uses attribute certificates to define authorization and delegation policies. Attributes are *connected* to privileges, e.g. The attribute *Friend* defined by *Bob* may be used to define who is granted permission to access some of his services, but can also be connected to other attributes, e.g. Attribute *Brother* can be connected to the attribute *Friend* in the sense that all privileges assigned to owners of *Friend* should also be assigned to owners of *Brother.*

When we talk about attributes, we normally only use the ID of the attribute but the manager or creator of the attribute should also be taken into account, this is why an attribute is defined as a tuple, $(AttributeManager, AttributeID)$. The attribute manager or source of attribute (SA), is the one which defines the attributes and its links with privileges and other attributes, so $(Bob, Friend)$ is the attribute *Friend* defined by *Bob*, which is different from $(Alice, Friend)$ although they have the same ID. An attribute is meaningless outside the scope of the SA, so $(Alice, Friend)$ is not meaningful in *Bob*'s authorizations. If we want to give meaning to an external attribute in our authorizations we should define what we call an attribute subscription (AS), which consists of a pair of attributes, eventually from different domains, in which the first one is said to be subscribed to the second one.

3. WTG VISUAL EDITOR

The previous section (section 2) details a theoretical model to implement controlled delegation. In order to apply the proposed model it is highly recommended to use a visual tool. This is why we implemented the WTG Visual Editor.

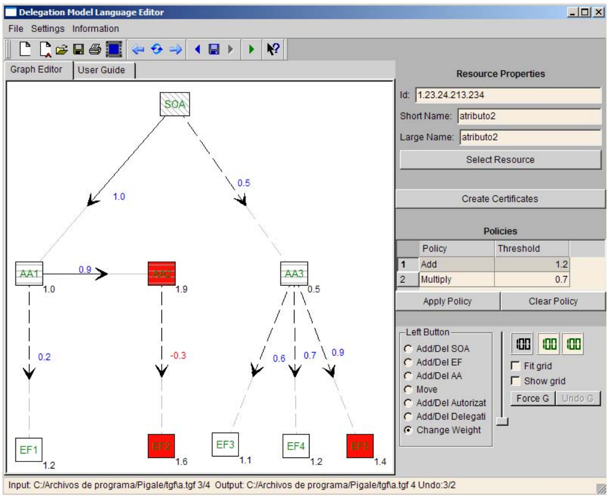
Several libraries are required for the deployment of the tool: OpenSSL, QT, GraphML, and PIGALE. An owner version of OpenSSL [11] is used to manage the attribute certificates related to the WTG graph. PIGALE, based on QT libraries, is the graphical support library used to draw graphs. The language selected to store the graphs is GraphML, based on XML.

There are two possible ways of creating a WTG instance. The first one is the usual way, i.e. by drawing elements in the editor and the second is by selecting a group of attribute certificates. These certificates store in the extension field the information needed to create the graph. The defined extension is named *WeightPathIdentifier* and its ASN.1 representation is detailed in the following paragraph. More detailed information about extension data and how use it, can be founded in [6, 10].

```
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
    }
```

Figure 1 details a graphical design using the visual editor. On the left (canvas) the user includes the actors (nodes) involved in the delegation and authorization sentences (edges). There are three types of actors: Source Authority (SoA), Attribute Authority (AA) and End User (EF). The description of these roles corresponds to those described in the ITU document [9]. **First, the user draws two actors and then draws the relationship between these actors,** the delegation (normal line) and authorization (slotted lines) sentences. Afterwards, the user assigns the weight to the sentences (numbers over edges).



Id: 1.23.24.213.234
Short Name: atributo2
Large Name: atributo2

	Policy	Threshold
1	Add	1.2
2	Multiply	0.7

Left Button: Add/Del SOA, Add/Del EF, Add/Del AA, Move, Add/Del Autorizat, Add/Del Delegati, [✓ Change Weight]

Input: C:/Archivos de programa/Pigale/tgf\a.tgf 3/4 Output: C:/Archivos de programa/Pigale/tgf\a.tgf 4 Undo:3/2

Figure 1: WTG instance in the Visual Editor

Template 1 Attribute Template

```
(deftemplate certificate
    (slot issuer (type SYMBOL))
    (slot holder (type SYMBOL))
    (slot attr_Resp (type SYMBOL))
    (slot attr_ID (type SYMBOL))
    (slot delegation (type INTEGER) (allowed-values 0 1) (default 0))
    (slot sign (type INTEGER) (allowed-values -1 1) (default 1))
    (slot weight (type FLOAT) (range 0.0 1.0) (default 1.0))
    (slot inferred (type SYMBOL) (allowed-values yes no) (default no)))
```

Once the WTG instance is designed, the visual tool offers two useful functionalities, automatic creation of the related Attributed Certificate and a simulation that returns those users which fullfil the authorization policies requisites. These functionalities are located in the right hand side of the application, *Create Certificates* and *Apply Policy* buttons respectively.

The automatic creation of the attribute certificates related to WTG allows the user to avoid this tedious and complex task. Moreover, the manual creation of the attribute certificates may result in mistakes in the data of the certificate extension.

The simulation functionality allows the user to simulate the performance of the designed instance of WTG. This allows us to establish whether users can do specific tasks and carry out appropiate changes to the instace until we obtain the desired result. The simulation engine is detailed in the following section.

4. DESIGN AND DESCRIPTION OF THE KNOWLEDGE BASED SYSTEM

As shown in the previous section (section 3), there is a need for a simulation engine to test the expected behaviour of WTG instances designed with the visual tool. To implement the simulation engine, we were inspired by knowledge based system. In particular, the selected language was CLISP.

The issuance of attributes is done by means of attribute certificates, with some extension, according to the ITU-T X.509 framework [9]. The *Issuer*, *Holder* and *Attribute_ID* are the corresponding fields from ITU-T X.509. The other fields are: The *Attribute_Manager*, as said previously is responsible for the attribute contained in the certificate. The *Delegation* field is 1 for delegation certificates and 0 for authorization certificates. The *Sign* field indicates whether the certificates assert attributes or deny them. Revocation is carried out by issuing a new negative certificate instead of by revoking existing certificates. The next field, *Weight*, is used to define a fine grain delegation and authorization. It offers the issuer the possibility of distinguishing between different authorization and delegation levels. In this way, different holders may receive different weights depending on the trust relationships between the issuer and the holder. The last field, *inferred* indicates whether the certificate has been inferred or not.

Then, an attribute certificate is described in CLIPS by using the following template,

As said before, attributes are described by an ID and its manager. In order to connect attributes belonging to different managers, we use attribute subscriptions. In CLIPS we use the following template to describe attribute subscriptions,

Template 2 Attribute Subscription

```
(deftemplate attr_subs
    (slot attrResp1 (type SYMBOL))
    (slot attrID1   (type SYMBOL))
    (slot attrResp2 (type SYMBOL))
    (slot attrID2   (type SYMBOL)))
```

If an attribute is subscribed to a second one, the second will be "translated" to the first one when an authorization decision is made in the domain of the first attribute manager, e.g. If $(Alice, Friend)$ is subscripted to $(Bob, Friend)$, Alice is stating that Bob's friends are also her friends, so she is delegating her attribute authority to Bob. To describe attributes we use both the "pair" notation, $(Alice, Friend)$, or the "dot" notation, $Alice.Friend$, in the text. More over, if we have two consecutive subscriptions, we can chain them. The purpose of this rule is merely to explore the relations of all the attributes in the systems.

Rule 4.1 Attribute Subscription Transitiveness

```
(defrule attr_subs_transitiveness
    (attr_subs (attrResp1 ?resp1) (attrID1 ?id1) (attrResp2 ?resp2) (attrID2 ?id2))
    (attr_subs (attrResp1 ?resp2) (attrID1 ?id2) (attrResp2 ?resp3) (attrID2 ?id3))
     =>
    (assert (attr_subs (attrResp1 ?resp1) (attrID1 ?id1) (attrResp2 ?resp3) (attrID2 ?id3))))
```

New attribute certificates can be derived from others using attribute subscriptions. As said before if attribute one is subscribed to attribute two, attribute two certificates have to be translated to attribute one certificates. Then, if there is an attribute certificate regarding attribute two, the same attribute, but regarding attribute one shall be inferred. The inferred certificate will have the inferred field set to yes.

Rule 4.2 Attribute subscription and certificate rule

```
(defrule attr_subs_certificate
    (attr_subs (attrResp1 ?resp) (attrID1 ?id) (attrResp2 ?resp2) (attrID2 ?id2))
    (certificate (issuer ?a) (holder ?b) (attrResp ?resp2)(attrID ?id2)
    (delegation ?del) (sign ?s) (weight ?w))
     =>
    (assert (certificate (issuer ?a) (holder ?b) (attrResp ?resp)
            (attrID ?id) (delegation ?del) (sign ?s) (weight ?w) (inferred yes))))
```

Delegation is effective only when the delegation policy holds, so the existence of delegation certificates is not enough. We use a specific delegation policy which checks whether the higher weight of all positive delegation credentials for the same holder is higher than the lower weight of all negative delegation credentials. Negative delegation certificates are intended to deny delegation, so if the "best" positive certificate has a weight greater than the "best" negative certificate, delegation becomes effective.

Rule 4.3 Delegation policy rule

```
(defrule delegation
 (certificate (issuer ?resp) (holder ?holder) (attrResp ?resp) (attrID ?id)
 (delegation 1) (sign 1) (weight ?w_plus))
 (forall (certificate (issuer ?resp) (holder ?holder) (attrResp ?resp)
 (attrID ?id) (delegation 1) (weight ?w_minus)) (test (<= ?w_minus ?w_plus)))
  =>
 (assert (delegation (holder ?holder) (attrResp ?resp) (attrID ?id))))
```

We represent paths of certificates by deriving an inferred certificate. Two credentials can be chained in the case where,

1. The first credential is a positive delegation credential, i.e. $sign = 1$ and $delegation = 1$.
2. The holder of the first credential has been delegated the corresponding attribute, i.e. $delegation(issuer1, resp, id)$ holds.
3. They are consecutive, i.e. the holder of the first credential is the issuer of the second one.
4. The attributes of the two credentials are the same.

As said before, the resulting path will be stored in a certificate with the inferred field set to yes. The weight of this virtual certificate will be the product of the weights of the two chained certificates. Consequently, the longer the chain is, the lower the weight will be. **The sign and delegation fields of the inferred certificates are taken from the second certificate.** The CLIPS rule used to chain certificates is,

Rule 4.4 Certificate chain rule

```
(defrule chain
    (delegation (holder ?holder1) (attrResp ?resp) (attrID ?id))
    (certificate (issuer ?issuer1) (holder ?holder1) (attrResp ?resp)
    (attrID ?id) (delegation 1) (weight ?x) (sign 1))
    (certificate (issuer ?holder1) (holder ?holder2) (attrResp ?resp)
    (attrID ?id) (delegation ?del) (weight ?y) (sign ?s))
    =>
    (assert (certificate (issuer ?issuer1) (holder ?holder2) (attrResp ?resp)
    (attrID ?id) (delegation ?del) (weight (* ?x ?y)) (sign ?s) (inferred yes))))
```

In an arbitrary environment, the two rules defined until now could be in conflict because of loops. In order to avoid this, we could prevent CLIPS from generating new inferred certificates in case a previous certificate exists with the same fields but with a greater weight. A path with a loop will have a lower weight than the same path without the loop, as the weight of the path is computed by multiplying the weights of the certificates in the path and these weights are between zero and one. Therefore paths with loops are naturally discarded, but in order to optimize the program, such a constraint could be explicitly defined at the begining of rule 4.4.

In [2], Toumas Aura analyzes the structure of delegation networks and somehow concludes that the effective delegation network is a Directed Acyclic Graph (DAG). For more information on DAG, see [4]. In DAGs there are no loops, in fact they correspond to partial order sets, so applying these rules to DAG is straightforward.

By using the delegation policy rule and the certificate chain rule, the authorization and delegation information is spread throughout the system. This has to be complemented by users defining authorization policies for their attributes. The matching of services, privileges and attributes is outside the scope of this paper. Once the attribute is considered valid for a requester, all the privileges linked to it will be available to him/her. Therefore, the manager of the attribute has to define authorization or *activation* policies for their attributes. Those policies consist of a lower bound for the lowest positive authorization weight, given that there are no negative authorization certificates. The higher the bound is, the more restrictive the policy will be. An authorization policy is defined using the following CLIPS template,

Template 3 Attribute activation policy

```
(deftemplate policy
    (slot attr_Resp (type SYMBOL))
    (slot attr_ID (type SYMBOL))
    (slot bound (type FLOAT) (range 0.0 1.0) (default 0.0)))
```

Then with the authorization rule, Figure 4.5, the effective granting of an attribute can be checked. This rule has a higher priority so inference stops as soon as the desired authorization predicates are inferred.

Rule 4.5 Authorization policy rule

```
(defrule authorization
    (declare (salience 10))
    (policy (attrResp ?resp) (attrID ?id) (bound ?x))
    (certificate (issuer ?resp) (holder ?holder)
         (attrResp ?resp)(attrID ?id) (delegation 0) (sign 1) (weight ?w))(test (> ?w 0))
    (not (exists (certificate (issuer ?resp) (holder ?holder)
    (attrResp ?resp) (attrID ?id) (delegation 0) (sign -1) (weight ?w))(test (> ?w 0))))
    (forall (certificate (issuer ?resp) (holder ?holder)
    (attrResp ?resp)(attrID ?id) (delegation 0) (sign 1) (weight ?w))(test (< ?x ?w)))
    =>
    (assert (authorization (holder ?holder) (attrResp ?resp) (attrID ?id))))
```

4.1 Goal oriented search

The previously presented set of rules will infer all the delegation and authorization relationships in the system. When trying to simulate the behavior of the set certificates defined with the visual tools, we have to take into account their size. If there are too many certificates, the inference using the previous rules will take too long. If we want to restrict the scope of the inference, i.e. focus on particular entities or certificates, we have to let CLIPS know our inference goal. CLIPS by itself does not manage goal oriented inference, so we have to include some details in the previous rules to make a CLIPS implementation usable. Those rules defined here, in this section, will replace the previous ones in the case where we want to CLIPS to work in goal oriented mode.

For each request, a tuple of the form

```
(authorization
     (holder Requester) (attrResp Attr_Resp) (attrID Attr_ID))
```

has to be found in the CLIPS facts set. This fact will be the inference goal.

Once the Goal is clear, we have to decide if we want the inference to be started by the user responsible for the attribute or by the requester, i.e. chaining certificates starting from the beginning or from the end. As the bidirectional search has been proven to be more efficient [1, 3], we try to start chaining certificates from both sides.

Attributes are chained backward and only attribute subscriptions leading to the requested one will be taken into account. To inform CLIPS about the desired inferred direction, we use facts of the form, *valid_issuerAttr_Resp*, *valid_holderRequester* and *valid_attrAttr_RespAttr_ID*. These facts will be used to refine the rules defined in the previous section so that new facts are near to the desired ones.

We start redefining rules about attributes. Rule 4.6 is restricted so the inferred attribute subscription is a subscription of a valid attribute with some other one. In Rule 4.7,

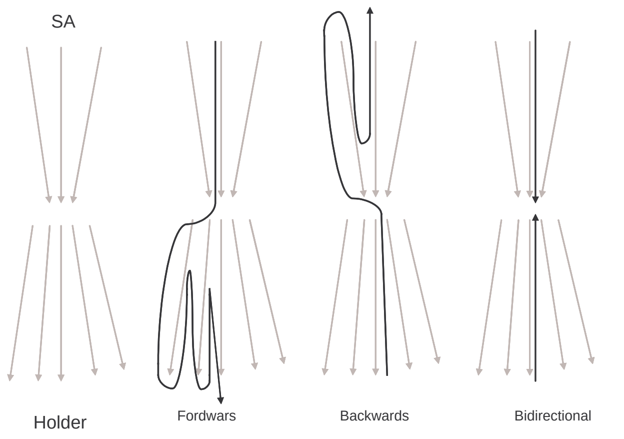


Figure 2: Different search algorithms in graphs

restriction consists of checking that the inferred certificate is a certificate about a valid attribute. The resulting rules are, Rule 4.6 and Rule 4.7.

Rule 4.6 Goal oriented attribute subscription transitiveness

```
(defrule attr_subs_transitiveness_goal
    (attr_subs (attrResp1 ?resp1) (attrID1 ?id1)(attrResp2 ?resp2)(attrID2 ?id2))
    (attr_subs (attrResp1 ?resp2) (attrID1 ?id2)(attrResp2 ?resp3)(attrID2 ?id3))
    (valid_attr ?resp1 ?id1)
     =>
    (assert (attr_subs (attrResp1 ?resp1) (attrID1 ?id1) (attrResp2 ?resp3) (attrID2 ?id3))))
```

Rule 4.7 Goal oriented attribute subscription and certificate rule

```
(defrule attr_subs_certificate_goal
    (valid_attr ?resp ?id)
    (attr_subs (attrResp1 ?resp) (attrID1 ?id) (attrResp2 ?resp2) (attrID2 ?id2))
    (certificate (issuer ?a) (holder ?b) (attrResp ?resp2)
    (attrID ?id2)(delegation ?del) (sign ?s) (weight ?w))
     =>
    (assert (certificate (issuer ?a) (holder ?b) (attrResp ?resp)
    (attrID ?id) (delegation ?del) (sign ?s) (weight ?w) (inferred yes))))
```

Together with these rules, we have to define a new rule, Rule 4.8, to derive new valid attributes, i.e. those which CLIPS will look for.

Rule 4.8 Valid attributes

```
(defrule valid_attribute
    (attr_subs (attrResp1 ?resp1) (attrID1 ?id1) (attrResp2 ?resp2) (attrID2 ?id2))
    (valid_attr ?resp1 ?id1)
     =>
    (assert (valid_attr ?resp2 ?id2)))
```

Lets now start redefining the rest of the rules. When chaining two certificates, either the issuer of the first one or the holder of the second one has to be a valid issuer or holder respectively and also the attribute has to be a valid attribute. This premise modifies Rule 4.4 to obtain Rule 4.9

Rule 4.9 Goal oriented certificate chain rule

```
(defrule chain_goal
  (delegation (holder ?holder1) (attrResp ?resp)(attrID ?id))
  (certificate (issuer ?issuer1) (holder ?holder1)
  (attrResp ?resp) (attrID ?id) (delegation 1) (weight ?x) (sign 1))
  (certificate (issuer ?holder1) (holder ?holder2)
   (attrResp ?resp)(attrID ?id) (delegation ?del) (weight ?y) (sign ?s))
  (or (valid_issuer ?issuer1) (valid_holder ?holder2))
  (valid_attr ?resp ?id)
    =>
  (assert (certificate (issuer ?issuer1) (holder ?holder2)
  (attrResp ?resp)(attrID ?id) (delegation ?del) (weight (* ?x ?y)) (sign ?s)(inferred yes))))
```

Moreover, when checking delegation effectiveness, only certificates about valid attributes have to be taken into account. This leads to a modification of Rule 4.3, which turns into Rule 4.10.

Rule 4.10 Goal Oriented delegation policy rule

```
(defrule delegation
 (certificate (issuer ?resp) (holder ?holder) (attrResp ?resp)
 (attrID ?id) (delegation 1) (sign 1) (weight ?w_plus))
 (forall (certificate (issuer ?resp) (holder ?holder)
 (attrResp ?resp)(attrID ?id) (delegation 1) (weight ?w_minus))
 (test (<= ?w_minus ?w_plus)))
 (valid_attr ?resp ?id)
 (valid_issuer ?resp)
  =>
 (assert (delegation (holder ?holder) (attrResp ?resp) (attrID ?id))))
```

In order to generate new valid issuers and holders we need to define these two rules,

Rule 4.11 Valid holder

```
(defrule valid_holder
    (certificate (issuer ?issuer) (holder ?holder) (attr_Resp ?resp) (attr_ID ?id))
    (valid_issuer ?issuer)
    (valid_attr ?resp ?id)
    =>
    (assert (valid_holder ?holder)))
```

Rule 4.12 Valid issuer

```
(defrule valid_issuer
    (delegation (holder ?holder) (attrResp ?resp) (attrID ?id))
    (valid_issuer ?resp)
    (valid_attr ?resp ?id)
    =>
    (assert (valid_issuer ?holder)))
```

With this modification, CLIPS follows the steps marked by the valid attributes, issuers and holders. In this way the computation is more efficient when asking a single request. So when the simulation is focuses on particular elements, this approach should be taken.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have implemented a simulation engine for a simplification of the WTG authorization model, and both a neutral and a goal oriented search algorithm have been implemented. Certificates are translated into CLIPS facts and then the program runs, providing answers to the authorization requests. The experimental results prove that using goal oriented algorithm is more efficient because the corresponding authorization predicate is inferred in fewer inference steps and less time. A formal proof for the algorithm is outside the scope of this work, in [6, 5] there is more information on the described algorithms.

This work complements WTG by allowing designers and security administrators to check if the defined certificates reflect the desired behavior of the system, before distributing the certificates.

6. REFERENCES

[1] Tuomas Aura Fast access control decisions from delegation certificate databases. Third Australasian Conference on Information Security and Privacy, pages 284–295, 1998.

[2] Tuomas Aura On the structure of delegation networks Proc. 11th IEEE Computer Security Foundations Workshop, 1998, IEEE Computer Society Press.

[3] Tuomas Aura Comparison of graph-search algorithms for authorization verification in delegation networks, In the proceedings of 2nd Nordic Workshop on Secure Computer Systems NORDSEC'97, Espoo, Finland, November 1997.

[4] Carlos Cotta, Jose M. Troya Analyzing Directed Acyclic Graph Recombination Computational Intelligence. Theory and Applications : International Conference, 7th Fuzzy Days Dortmund, Germany, October 1-3, 2001, Proceedings.

[5] Isaac Agudo, Javier Lopez and Jose A. Montenegro Graphical Representation of Authorization Policies for Weighted Credentials In 11th Australasian Conference on Information Security and Privacy. (ACISP'06), pp. 383-394. LNCS 4058, Springer. Melbourne, Australia. July 2006.

[6] Isaac Agudo, Javier Lopez and Jose A. Montenegro. A representation model of trust relationships with delegation extension. In *3rd International Conference on Trust Management, iTrust 2005*, volume 3477 of *Lecture Notes in Computer Science*, pages 116 – 130. Springer, 2005.

[7] Yan Zhang Chun Ruan, Vijay Varadharajan. Logic-based reasoning on delegatable authorizations. In *Foundations of Intelligent Systems : 13th International Symposium, ISMIS*, 2002.

[8] Ninghui Li, John C. Mitchell, and William H. 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.

[9] ITU-T X.509, ISI/IEC 9594-8, Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks. 08/2005.

[10] Isaac Agudo, Javier Lopez, Jose A. Montenegro A Graphical Delegation Solution for X.509 Attribute Certificates ERCIM News. SPECIAL THEME: Security and Trust Management No. 63, Octubre 2005, pp. 33-34. ISSN: 0926-4981

[11] Jose A. Montenegro, Fernando Moya A practical approach of X509 Attribute Certificate Framework as support to obtain Privilege Delegation 1st European PKI Workshop: Research and Applications. Isla de Samos, Grecia. Junio 2004 LNCS 3093, Springer-Verlag, pp. 160-172. ISBN 3-540-22216-2