UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

INTERACTIVE ACCESS CONTROL IN AUTONOMIC
COMMUNICATION

Hristo Koshutanski and Fabio Massacci

October 2004

Technical Report # DIT-04-094

Interactive Access Control in Autonomic Communication

Hristo Koshutanski and Fabio Massacci

{hristo, massacci}@dit.unitn.it

Dip. di Informatica e Telecomunicazioni - Univ. di Trento

via Sommarive 14 - 38050 Povo di Trento (ITALY)

Abstract

Autonomic Communication is a new paradigm for dynamic network integration. An Autonomic Network crosses organizational and management boundaries and is provided by entities that see each other just as business partners. Policy-based network access and management already requires a paradigm shift in the access control mechanism: from identity-based access control to trust management and negotiation, but this is not enough for cross organizational autonomic communication. For many services no autonomic communication partner may guess a priori what will be sent by clients and clients may not know a priori what credentials are demanded for completing a service requiring the orchestration of many different autonomic nodes.

To solve this problem we propose to use interactive access control for autonomic communication: servers should be able to get back to clients asking for missing credentials, whereas the latter may decide to supply or decline requested credentials and so on until a final decision is made. This proposal is grounded in a formal model on policy-based access control using abduction. We identify the key algorithm for interactive access and show its correctness. The Web Services-based implementation that we have developed is also sketched.

Keywords: Interactive Access Control, Adaptive Access Control, Self-Managing Systems, Security Management, Autonomic Communication, Controlled Disclosure, Credential-Based Systems, Internet Computing, Logics for Access Control.

CONTENTS

I. INTRODUCTION

Controlling access to services is a key aspect of networking and the last few years have seen the domination of policy-based access control. Indeed, the paradigm is broader than simple access control and one may speak of *policy-based network self-management* (See [37], [27] or the IEEE Policy Workshop series for examples). The intuition is that actions of nodes "controlling" the communication are automatically derived from policies. The nodes look at events and requests presented to them, evaluate the rules of their policies according those new facts and derive the actions [37], [38]. Policies can be "simple" `iptables` configuration rules for Linux firewalls[1] or complex logical policies expressed in languages such as Ponder [8].

Autonomic Communication adds new challenges: a truly autonomic network is born when nodes are no longer within the boundary of a single enterprise which could deploy its policies on each and every node and guarantee interoperation. Services on the border are then fairly limited to existing industry standard (again to guarantee interoperation). In an autonomic network, nodes are partners that offer services and lightly integrate their efforts into one (hopefully coherent) network. This cross enterprise scenario poses novel security challenges with aspects of both trust management systems and workflow security.

From trust management systems [41], [10], [24] it takes the credential-based view. Since access to network services is offered by autonomic nodes on their own to potentially unknown clients, the decision to grant or deny access can only be made on the basis of the credentials sent by the client. In contrast with trust management systems, we have a continuous process and thus a notion of assignment of permissions to credentials that must look beyond the single access decision.

From workflow access control systems (see e.g. [2], [4], [12], [18]) we borrow all classical problems such as dynamic assignment of roles to users, dynamic separation of duties, and assignment of permissions to users according the least privilege principles. In contrast with workflow security management schemes, we can no longer assume that the enterprise will assign tasks and roles to users (its employees) in such a way that makes the overall flow possible w.r.t. its security constraints. The reason is that the enterprise itself no longer exists.

In an autonomic communication scenario a client might have all the necessary credentials to

---

[1]See http://www.netfilter.org/.

access a service but may simply not know it. Equally, it is unrealistic to assume that servers will publish their security policies on the web so that a client can do a policy combination and evaluation themselves. So, it should be possible for a server to ask a client on the fly for additional credentials and the same may disclose them or decline to provide them. The server can then re-evaluate the client request at the light of the new submitted credentials, and interate the process until a final decision (grant or deny is made). We call this modality *interactive access control*.

Part of these challenges can be solved by using policy-based self-management of networks but not all of them. Indeed, if we abstract away the details of the policy implementation, we can observe that the only reasoning service that is actually used by policy-based self-management approaches is *deduction*: given a policy and a set of additional facts find out all consequences (actions or obligations) of the policy and the facts. We simply look whether granting the request can be deduced from the policy and the current facts. Policies can be different [3], [24], [5], [4] but the kernel reasoning service is the same.

A. The Contribution of this paper

We claim that autonomic communication needs at least another reasoning service: *abduction* [36]. Loosely speaking, we could say that abduction is deduction in reverse: given a policy and a request to access a network service, we want to know which are the credentials/events that would grant access. Logically speaking, we want to know whether there is a (possibly minimal) set of facts that could be added to the policy so that the request can be deduced from it.

If we look again at our intuitive description of interactive access control, it is immediate to realize that abduction is a core service needed by policy-based autonomic servers to get back to autonomic clients. Indeed, we might even want the same service to run on both client and server sides whenever the client also requires some evidence from the server in order to establish trust before disclosing his own credentials.

Here, we present a framework for reasoning about interactive access control for autonomic communication that tries to answer these challenges and that is grounded in a formal theory by using logic-based policies.

We start by presenting the basic framework (§.II) and by introducing a running example to make discussion more concrete (§.III). Next, we introduce the formal syntax for the formal model

(§.IV), identify the different formal reasoning services – deduction vs abduction (§.VI) – that characterize the problem. At this stage we have all necessary material to introduce the interactive access control algorithm (§.VII) and show that it delivers its promises (§.VIII): a client can only get access if he has submitted the right credentials, and (most important) a client who has the right set of credentials and who is willing to send them to the server, will not be left stranded in our autonomic network. Finally, we briefly sketch our implementation based on web services (§.IX) and conclude the paper with a brief discussion of related work.

II. THE BASIC FRAMEWORK

Using Datalog and logic programs for representing and reasoning about access control is customary in computer security [3], [24], [5], [4] and this work is no exception. Our formal model for reasoning about access control is based variants of Datalog with the stable model semantics and combines in a novel way a number of features:

- logic for trust management by Li et al. [24];
- logic for workflow access control by Bertino et al. [4];
- logic for disclosure and access control by Bonatti and Samarati [5];
- some ideas from trust negotiation by Yu, Winslett and Seamons [44].

We consider the view of a single partner since we cannot assume sharing of policies between partners.

In our framework each partner has a *security policy for access control* $\mathcal{P}_{\mathcal{A}}$ and a *security policy for disclosure control* $\mathcal{P}_{\mathcal{D}}$. The policy for access control is used for making decision about usage of all web services offered by a partner. The policy for disclosure control is used to decide credentials whose need can be potentially disclosed to a client. In other words, $\mathcal{P}_{\mathcal{A}}$ protects partner's resources by stipulating what credentials a requestor must satisfy to be authorized for a particular resource while, in contrast, $\mathcal{P}_{\mathcal{D}}$ defines which credentials among those occuring in $\mathcal{P}_{\mathcal{A}}$ are disclosable so, if needed, can be demanded from the requestor.

Other approaches (e.g. Yu et al [44, page 38] and Winsborough et al [43, page 12] examples) offers the two policies into one. However, we believe that the separation of access and disclosure policies is useful for practical reason in spite of the additional complication that the two-policies system requires for evaluation. The double query system is immaterial to the human administrator who might simply buy a faster machine. In contrast, the specification of policies is normally

done by humans and is a costly and error prone process. The integration of disclosure and access policy into one policy would imply that a change in the disclosure policy requires modification to the access policy which might have been unchanged. Furthermore the separation of access and disclosure policies allows for separation of duties among administrators: one administrator can modify the access policy and another the disclosure policy.

We keep a set of *active credentials* $\mathcal{C}_\mathcal{P}$ that have been presented by the client in past interactions within the same service session and a set of *declined credentials* $\mathcal{C}_\mathcal{N}$ also compiled from the client's past interactions. So, to execute a service of the fragment of a partner a user will submit a set of *presented credentials* $\mathcal{C}_r$ and a *service request* r. In the same context, $\mathcal{C}_\mathcal{N}$ is computed as a difference between the missing credentials $\mathcal{C}_\mathcal{M}$, the client was asked in the last interaction, and the credentials presented in the current step, namely $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{M} \setminus \mathcal{C}_r$.

III. A Running Example

Let us assume that we have a Planet-Lab shared network between the University of Trento and Fraunhofer institute in Berlin in the context of the E-NEXT network. For the sake of simplicity let us also assume that there are three main access types to the resources: *disk* – (read) access to data residing on the Planet-Lab machines; *run* – (execute) access to data and possibility to run processes of the machines; and *configure* – (addService) including the previous two types of accesses plus the possibility of configuring network services on the machines.

We also suppose that all Planet-Lab credentials (certificates) are signed and issued by trusted authorities and that the validation of these credentials is performed before the actual access control process. This can be done by plugging in any standard Privilege Management Infrastructure (PMI).

We can now formalize our running example. Fig. 1 shows the joint hierarchy, the roles and their relative hierarchy. The partial order of roles is indicated by arcs, where higher the role in the hierarchy, more powerful it is. A role dominates another role if it is higher in the hierarchy and there is a direct path between them.

The access policy says:

– Rules (1) and (2) give access to the shared network content to everybody from the University of Trento and Fraunhofer institute, regardless the IP and roles at these institutions.
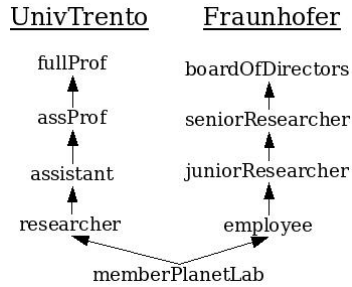
Fig. 1.   Joint Hierarchy Model

– Rules (3) and (4) allow access from those machines that are internal for the two institutions and located in the internal LANs (dedicated machines only for Planet-Lab acccess) distinguished by their fixed IPs.

– Rule (5) relaxes the previous two and allows access from any place of the institutions provided users declare their ID and present some role-position certificate of their organization or at least a Planet-Lab membership credential.

– Rules (6) and (7) say that if a user has got a light access and is at minimum researcher at UnivTrento or junior researcher at Fraunhofer, it has additional rights.

– Rules (8) and (9) relaxed associate professors and senior researchers from the fact that they can access the network resources from any place they want under the respective country domain (e.g. home, other universites, etc).

– Rules (10) and (11) give full access, from any place of the world, only to members of board of directors and to full professors.

The disclosure policy says:

– Rules (1) and (2) disclose the need for the client to declare its ID if the same comes from an authorized network of the respective organizations;

– Rule (3) discloses the need for Planet-Lab membership credential if the client has already declared its ID;

– Rule (4) discloses (upgrades) the need of a higher role-position credential.

Now, examine the case in which a senior researcher at Fraunhofer institute wants to have access to the system from his home place (decided to work from home) presenting its employee

Role: $R_i \succ$ Role: R_j when role Role: R_i dominates role Role: R_j.

Role: $R_i \succ_{\text{WebServ:}S}$ Role: R_j when role Role: R_i dominates, just for service WebServ: S, the role Role: R_j.

assign $(P, \text{WebServ:}S)$ when an access to the service WebServ: S is granted to P. P can be either a Role: R or User: U.

forced $(P, \text{WebServ:}S)$ when access the service WebServ: S must be forced to P. P can be either a Role: R or User: U.

(a) Predicates for assignments to Roles and Services

declaration $(\text{User:}U)$ it is a statement by the User: U for its identity.

credential $(\text{User:}U, \text{Role:}R)$ when User: U has a credential activating Role: R.

credentialTask $(\text{User:}U, \text{WebServ:}S)$ when User: U has the right to access WebServ: S.

(b) Predicates for Credentials

running $(P, \text{WebServ:}S, \text{number:}N)$ when the Nth activation of service S is executed by P.

abort $(P, \text{WebServ:}S, \text{number:}N)$ if the Nth activation of service S within a workflow aborts.

success $(P, \text{WebServ:}S, \text{number:}N)$ if the Nth activation of service S within a workflow successfully executes.

grant $(P, \text{WebServ:}S, \text{number:}N)$ if the Nth request of service S has been granted

deny $(P, \text{WebServ:}S, \text{number:}N)$ if the Nth request of service S has been denied.

(c) Predicates describing the current status of services

Fig. 2.   Predicates used in the model

certificate. So, according to rules (8) and (9) the system should deny access. But is it always the behaviour we want from the system? Shall we leave him harassing of being idle for the whole day simply because he did not know or just has forgotten that access from home to the system needs another certificate? The full formalization of the example we will see in §VII.

IV. SYNTAX

For the syntax we build upon [4], [5], [24]. We have three disjoint sets of constants: one for users identifiers denoted by User : U; one for roles denoted by Role : R; and one for services denoted by WebServ: S.

The predicates can be divided into three classes: predicates for assignments of users to roles and services (Fig. 2a), predicates for credentials (Fig. 2b), and predicates describing the current status of the system (Fig. 2c). The last class of predicates keeps track on the main activities done by users and services.

Furthermore, for some additional workflow constraints we need to have some meta-level predicates that specify how many statements are true. We use here a notation borrowed from Niemela smodels system, but we are substantially using the count predicates defined by Das [9]:

$n \leq \{X. Pr\}$ where n is a positive integer, X is a set of variables, and Pr is a predicate, so that intuitively $n \leq \{X. Pr\}$ is true in a model if at least n instances of the grounding of X variables in Pr are satisfied by the model. The $\{X. Pr\} \leq n$ is the dual predicate.

We assume additional comparison predicates (for instance for equality or inequalities) or some additional monadic predicates for instance to qualify services or users.

We note here that the model, presented in the this section, can be adapted to *any* generic policy framework. The information we need from the underlying policy model is shown in Figure 2(a, b) and that infromation can be found in (extracted from) most policy languages.

V. SEMANTICS

The semantics is based on normal logic programs [1]. These are sets of *rules* of the form:

$$A \quad \leftarrow \quad B_1, \ldots, B_n, \ not \, C_1, \ldots, \ not \, C_m \tag{1}$$

where A, B_i and C_i are (possibly ground) predicates among those described in §IV. A is called the *head* of the rule, each B_i is called a *positive literal* and each $not \, C_j$ is a *negative literal*, whereas the conjunction of the B_i and $not \, C_j$ is called the *body* of the rule. If the body is empty the rule is called a *fact*. A normal logic program is a set of rules.

In our framework, we also need *constraints* that are rules with an empty head.

$$\leftarrow \quad B_1, \ldots, B_n, \ not \, C_1, \ldots, \ not \, C_m \tag{2}$$

One of the most prominent semantics for normal logic programs is the *stable model semantics* proposed by Gelfond and Lifschitz [11] (see also [1] for an introduction). The intuition is to interpret the rules of a program P as constraints on a solution set S (a set of ground atoms)

for the program itself. So, if S is a set of atoms, rule (1) is a constraint on S stating that if all B_i are in S and none of C_j are in it, then A must be in S. A constraint (2) is used to rule out from the set of acceptable models the situation in which B_i are true and all C_j are false is not acceptable.

We now consider ground rules, i.e. rules where atoms do not contain variables.

Definition 5.1: The *reduct* P^S of a ground logic program P with respect to a set of atoms S is the definite program obtained from P by deleting:

1) each rule that has a negative literal $not\, C$ in its body with $C \in S$;

2) each negative literal in the bodies of the remaining rules.

The reduct P^S is a definite logic program. Let $M(P^S) = M_{Ps}$ be the semantics of the definite logic program P^S, i.e. its minimal model.

Definition 5.2: A set of atoms S is a stable model of a normal logic program P iff $S = M(P^S)$.

A program can have none, one or many stable models. The definition of stable models captures the two key properties of solution sets of logic programs.

1) Stable models are minimal: a proper subset of a stable model is not a stable model.

2) Stable models are grounded: each atom in a stable model has a justification in terms of the program, i.e. it is derivable from the reduct of the program with respect to the model.

Though this definition of stable models in terms of fix points is non-constructive there are constructive definitions [1] and systems [32], [23] that can cope with ground programs having tens of thousands of rules.

Logic programs with variables can be given semantics in terms of stable models.

Definition 5.3: The stable models of a normal logic program P with variables are those of its ground instantiation P_H with respect to its Herbrand universe[2].

If logic programs are function free, then an upper bound on the number of instantiations is rc^v, where r is the number of rules, c the number of the constants, and v the upper bound on the number of distinct variables in each rule.

---

[2]Essentially, we take all constants and functions appearing in the program and combine them in all possible ways. This yields the Herbrand universe. Those terms are then used to replace variables in all possible ways thus building its ground instantiation.

Definition 5.4 (Logical Consequence and Consistency): Let P be a logic program and L be a (positive or negative) ground literal. L is a *logical consequence* of P ($P \models L$) if L is true in every stable model of P. P is *consistent* ($P \not\models \perp$) if there is a stable model for P.

Definition 5.5 (Security Consequence): A request r is a security consequence of a policy $\mathcal{P}_\mathcal{A}$ if (i) $\mathcal{P}_\mathcal{A}$ is logically consistent and (ii) r is a logical consequence of $\mathcal{P}_\mathcal{A}$.

Definition 5.6 (Abduction): Let P be a logic program, H a set of predicates (called hypothesis, or abducibles), L a (positive or negative) ground literal, and $\prec$ a partial order (p.o.) over subsets of H. A *Cautious solution* of the abduction problem is a set of ground atoms E such that

 (i) $E \subseteq H$,

 (ii) $P \cup E \models L$,

(iii) $P \cup E \not\models \perp$,

(iv) any set $E' \prec E$ does not satisfy all conditions above.

Traditional p.o.s are subset containment or set cardinality. Other solutions are possible with orderings over predicates.

VI. Logical Model

In this section we give formal definitions of the security policies introduced informally at the beginning of the paper.

Definition 6.1 (Access Policy): An *access control policy* $\mathcal{P}_\mathcal{A}$ is a logic program over the predicates defined in Section IV in which

 (i) no credential and no execution atom can occur in the head of a rule,

 (ii) role hierarchy atoms occur as facts,

An access request r is a ground instance of an $\mathsf{assign}\,(\mathsf{User}\!:\!U, \mathsf{WebServ}\!:\!S)$ predicate.

In contrast to the proposal by Bertino et al. [4] for workflows, we don't need any special rule for determining which services cannot be executed and which services must be executed by a specific user or role. Logic constraints guarantee the same result.

Definition 6.2 (Disclosure Policy): A *disclosure policy* $\mathcal{P}_\mathcal{D}$ is a logic program in which no role hierarchy atom and no execution atom can occur in the head of a rule.

Following is the full formalization of the running example introduced in §III. There is a preprocessing step that validates and transforms certificates to predicates suitable for the formal model – $\mathsf{credential}\,(\mathsf{User}\!:\!U, \mathsf{Role}\!:\!R)$. The new predicate used in the example is

Access Policy:

(1) assign $(*, request(read)) \leftarrow$ authNetwork $(*, *.unitn.it)$.

(2) assign $(*, request(read)) \leftarrow$ authNetwork $(*, *.fraunhofer.de)$.

(3) assign $(*, request(execute)) \leftarrow$ authNetwork $(193.168.205.*, *.unitn.it)$.

(4) assign $(*, request(execute)) \leftarrow$ authNetwork $(198.162.45.*, *.fraunhofer.de)$.

(5) assign $(User, request(execute)) \leftarrow$ assign $(User, request(read))$, declaration $(User)$,
        credential $(User, Role)$, $Role \succeq memberPlanetLab$.

(6) assign $(User, request(addService)) \leftarrow$ assign $(User, request(execute))$, declaration $(User)$,
        credential $(User, Role)$, $Role \succeq researcher$.

(7) assign $(User, request(addService)) \leftarrow$ assign $(User, request(execute))$, declaration $(User)$,
        credential $(User, Role)$, $Role \succeq juniorResearcher$.

(8) assign $(User, request(addService)) \leftarrow$ authNetwork $(*, *.it)$, declaration $(User)$,
        credential $(User, Role)$, $Role \succeq assProf$.

(9) assign $(User, request(addService)) \leftarrow$ authNetwork $(*, *.de)$, declaration $(User)$,
        credential $(User, Role)$, $Role \succeq seniorResearcher$.

(10) assign $(User, request(addService)) \leftarrow$ authNetwork $(*, *)$, declaration $(User)$,
        credential $(User, Role)$, $Role \succeq fullProf$.

(11) assign $(User, request(addService)) \leftarrow$ authNetwork $(*, *)$, declaration $(User)$,
        credential $(User, Role)$, $Role \succeq boardOfDirectors$.

Disclosure Policy:

(1) declaration $(User) \leftarrow$ authNetwork $(*, *.unitn.it)$.

(2) declaration $(User) \leftarrow$ authNetwork $(*, *.fraunhofer.de)$.

(3) credential $(memberPlanetLab, User) \leftarrow$ declaration $(User)$.

(4) credential $(RoleX, User) \leftarrow$ credential $(RoleY, User)$, $RoleX \succ RoleY$.

Fig. 3.   Proxy Access and Release Policies for an Online Library

authNetwork $(IP, DomainName)$. It is a tuple with first argument the IP address of the autho-

rized network endpoint (the client's machine) and the second argument the domain name where

the IP address comes from. Figure 3 shows the complete formalization of the policies.

Example 1 (Policy Constraints): Consider a security policy in which having a credential for

the role $accountant$ is incompatible with the assignment of any role $manager$, and that the

execution of a service $phoneCall$ from user $billG$ requires that the service $answer$ must be

executed by anybody having the role $headOfStaff$. The following rules guarantees the desired

behavior:

$\leftarrow$ credential (User : U, Role : $accountant$), assign (User : U, Role : $manager$).

forced (Role : $headOfStaff$, WebServ : $answer$)$\leftarrow$ running (User : $billG$, WebServ : $call$, number : N).

assign (P, WebServ : S)$\leftarrow$ forced (P, WebServ : S).

∎

Example 2 (Access Policy and Separation of Duty Constraints): Consider an e-stock portal where we have roles associated to services as follows: role eSeller – for selling shares and bonds on the floor; role eBuyer – for buying shares and bonds; role eAdvisor – used by accredited consultants to sell their advice to other customers of the portal. Then examine the case where one could send the eAdvisor credential to the service publishing advisories and suggest to sell shares, and at the same time the eBuyer credential to the service hosting bids.

In such situations we can define separation of duty rules:

$customer$(eSeller)$\leftarrow$.

$customer$(eBuyer)$\leftarrow$.

$\leftarrow$assign (User : U, Role : R_1), $customer$(R_1), assign (User : U, Role : eAdvisor).

The access control rule on reviewing selling bids is the following:

assign (User : U, WebServ : S) $\leftarrow$ credential (User : U, Role : R), assign (Role : R, WebServ : S).

assign (Role : R, WebServ : $reviewSell$) $\leftarrow$ Role : R $\succ$ Role : eSeller.

∎

As mentioned, we will use the disclosure policy $\mathcal{P}_\mathcal{D}$ to decide which missing credentials are to be asked from the client.

Example 3 (Disclosure Policy): Considering again the access policy in Example 2. A possible (part of) the disclosure policy $\mathcal{P}_\mathcal{D}$ could be:

credential (User : U, Role : $eUser$)$\leftarrow$declaration (User : U).

credential (User : U, Role : $eSeller$)$\leftarrow$credential (User : U, Role : $eUser$).

credential (User : U, Role : $eSellerVIP$)$\leftarrow$credential (User : U, Role : $eSeller$).

The second rule says: to reveal the need for a *eSeller* credential there should be already a credential attesting the client as a valid user (Role : eUser) of the system. ∎

So, the request assign (User : fm, WebServ : $reviewSell$) together with

credential (User : fm, Role : $eUser$) and declaration (User : fm) will yield a counter request –

credential $(\mathsf{User} \colon fm, \mathsf{Role} \colon eSeller)$ – specifying the need for additional privileges necessitated to get the service.

Note that the need for a credential attesting the role *eSellerVIP*, disclosed together with *eSeller*, should not be considered as a potential output by the system because the "intuition" says that *eSeller* is enough.

Remark 1 (Notion of Minimality): The choice of the partial order has a major impact in presence of complex role hierarchies. The "intuitive" behavior of the abduction algorithm for the extraction of the *minimal* set of security credentials is not guaranteed by the straightforward interpretation of H (abducibles) as the set of credentials and by the set cardinality or set containment orderings.

Consider the following program:

$$\mathsf{Role} \colon r_2 \succ \mathsf{Role} \colon r_1 \leftarrow \,.$$

$$\mathsf{assign} \, (\mathsf{User} \colon U, \mathsf{WebServ} \colon ws) \leftarrow \mathsf{credential} \, (\mathsf{User} \colon U, \mathsf{Role} \colon R), \mathsf{Role} \colon R \succ \mathsf{Role} \colon r_1.$$

Request $\mathsf{assign} \, (\mathsf{User} \colon fm, \mathsf{WebServ} \colon ws)$ has two $\subseteq$-minimal solutions:

$$\{\mathsf{credential} \, (\mathsf{User} \colon fm, \mathsf{Role} \colon r_1)\} \,, \{\mathsf{credential} \, (\mathsf{User} \colon fm, \mathsf{Role} \colon r_2)\}$$

Yet, our intuition is that the first should be the minimal one.

So, we need a more sophisticated partial order rather than set cardinality or set containment (rf. Remark 1). For example, we could stipulate that $E \preceq E'$ is such that for all credentials $c \in E$ there is a credential $c' \in E'$ where $c = c'$, we can revise it so that $E \prec E'$ if for $c \in E$ there is a credential $c' \in E'$ where c' is identical to c except that it contains a role R' that dominates the corresponding role R in c. This p.o. generates the "intuitive" behavior of the abduction algorithm.

An effective approximation of the above criterion, currently implemented in out prototype, is to include extra information in credentials from the hypotheses (abducibles), specifying the position of a role in the role lattice hierarchy. We called the extra information *role weight* indicating the *highest possible* position of a role in the lattice. The counting is bottom-up starting from the most bottom role(s) and reflects the number of roles the current one dominates with the direct path between them. So, if a role occurs in two different paths in the lattice, for example, once with weight 5 and once with 4 we select 5 as the role weight. Then it is easy to select those sets with lowest possible role weights – addressing in this case the least privileged principle.

This is why we need the highest possible value because we can accurately compute the minimal privilege of the maximal importance of a role. After slecting the minial sets (there could be more than one equally minimal) we perform a minimal set cardinality filtering to choose the final one. After having obtained the set of missing credentials, we drop this extra information from the set that is to be sent back to the client.

VII. INTERACTIVE ACCESS CONTROL

In this section we show how the various notions that we have seen so far can be combined into a comprehensive authorization mechanism. An authorization system receives a request r, processes it according to the access control algorithm and eventually takes a decision. A decision may have involved interactions and so we also keep track of the current set of active credentials $\mathcal{C}_\mathcal{P}$.

Since a client must have all relevant credentials (if required) for getting access to a service, one could borrow mechanisms for certificate chain discovery from [26], [7]. It is essential and important for any trust management system.

Once again it is worth noting that this view is partial as we only focus on the knowledge of one single autonomic node: there is no authorization domain crossing partnerships.

Our interactive access control solution is shown in Figure 4. The intuition behind the interactive access control algorithm is the following. Initially a client will submit a set of credentials $\mathcal{C}_r$ and a service request r (step 1). $\mathcal{C}_r$ is optional and so initially may be also an empty set. Once the client has initiated a session, the interactive algorithm is strated with internal input: the policy for access control $\mathcal{P}_\mathcal{A}$ and policy for disclosure control $\mathcal{P}_\mathcal{D}$.

Then the client's profile $(\mathcal{C}_\mathcal{P}, \mathcal{C}_\mathcal{N})$ is update as: active credentials $\mathcal{C}_\mathcal{P}$ are updated with the newly presented credentials $\mathcal{C}_r$; and declined credentials are updated as a set difference of what the client was asked in the last interaction $(\mathcal{C}_\mathcal{M})$ minus what he presents in the current interaction $(\mathcal{C}_r)$. Steps 2 and 3. After the client's profile is updated, the algorithm checks whether the request r is granted by $\mathcal{P}_\mathcal{A}$ according to the client's set of active credentials $\mathcal{C}_\mathcal{P}$ (step 4).

In the case of denial (step 5c), the algorithm computes all credentials disclosable from $\mathcal{P}_\mathcal{D}$ according to $\mathcal{C}_\mathcal{P}$ and from the resulting set removes all already declined and already presented credentials. In this case we avoid dead loops of asking something already declined or presented. Then we compute (using abduction reasoning) all possible subsets of $\mathcal{C}_\mathcal{D}$ that are consistent with

Global vars: $\mathcal{C}_{\mathcal{N}}$, $\mathcal{C}_{\mathcal{P}}$, $\mathcal{C}_{\mathcal{M}}$; Initially $\mathcal{C}_{\mathcal{N}} = \mathcal{C}_{\mathcal{P}} = \mathcal{C}_{\mathcal{M}} = \emptyset$;

Internal input: $\mathcal{P}_{\mathcal{A}}$, $\mathcal{P}_{\mathcal{D}}$;

Output: grant/deny/ask($\mathcal{C}_{\mathcal{M}}$);

1) **client's input**: $\mathcal{C}_r$ and r,
2) update $\mathcal{C}_{\mathcal{P}} = \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_r$,
3) update $\mathcal{C}_{\mathcal{N}} = (\mathcal{C}_{\mathcal{N}} \cup \mathcal{C}_{\mathcal{M}}) \setminus \mathcal{C}_r$, where $\mathcal{C}_{\mathcal{M}}$ is from the last interaction,
4) verify whether the request r is a security consequence of the policy access $\mathcal{P}_{\mathcal{A}}$ and presented credentials $\mathcal{C}_{\mathcal{P}}$, namely $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \models r$ and $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \not\models \bot$,
5) <u>if</u> the check succeeds <u>then</u> return ***grant*** <u>else</u>

   a) compute the set of *disclosable credentials* $\mathcal{C}_{\mathcal{D}}$ as
   $$\mathcal{C}_{\mathcal{D}} = \{c \mid c \text{ credential that } \mathcal{P}_{\mathcal{D}} \cup \mathcal{C}_{\mathcal{P}} \models c\} \setminus (\mathcal{C}_{\mathcal{N}} \cup \mathcal{C}_{\mathcal{P}}) \ ,$$

   b) use abduction to find a minimal set of missing credentials $\mathcal{C}_{\mathcal{M}} \subseteq \mathcal{C}_{\mathcal{D}}$ such that both $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{M}} \models r$ and $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{M}} \not\models \bot$,

   c) <u>if</u> no such set exists <u>then</u> return ***deny*** <u>else</u>

   d) return ***ask(*** $\mathcal{C}_{\mathcal{M}}$ ***)*** and iterate the process.

Fig. 4.   Interactive Access Control Algorithm

the access policy $\mathcal{P}_{\mathcal{A}}$ and, at the same time, grant r. Out of all these sets (if any) the algorithm selects the minimal one.

Example 4 (A Running Scenario): A senior researcher at Fraunhofer institute FOKUS wants to reconfigure an online service for paper submissions of a workshop. The service is part of a big management system hosted at the University of Trento's network that is part of Planet-Lab. So, for doing that, at the time of access, he presents his employee membership token, issued by a Fraunhofer certificate authority, presuming that it is enough as a potential customer.

Formaly speaking, the request comes from a domain *fokus.fraunhofer.de* with credential for Role : *employee* together with a declaration for a user ID, *John Milburk*. The set of credentials

is:

$$\{\text{authNetwork}\,(198.162.193.46,\, fokus.fraunhofer.de),$$
$$\text{credential}\,(JohnMilburk, employee),$$
$$\text{declaration}\,(JohnMilburk)\}$$

So, according to the access policy the credentials are not enough to get full access and the request would be denied (rf. rule 6 of Policy Access in Figure 3). Then, following the algorithm (step 5a in Figure 4) it is computed the set of disclosable credentials from the disclosure policy and the user's set of active credentials. In our case, $\mathcal{C}_{\mathcal{P}}$ is the set of credentials mentioned above. The algorithm computes $\mathcal{C}_{\mathcal{D}}$ as the need of all roles higher in position than Role: *employee* (rf. Figure 3 rule 4 of Disclosure Policy):

$$\{\text{credential}\,(\text{User}: JohnMilburk, \text{Role}: junior Researcher),$$
$$\text{credential}\,(\text{User}: JohnMilburk, \text{Role}: senior Researcher),$$
$$\text{credential}\,(\text{User}: JohnMilburk, \text{Role}: boardOf Directors)\}$$

The next step, abduction (Figure 4 step 5b), computes the minimal set of credentials, out of those, that satisfies the request. The resulting set is $\{\text{credential}\,(\text{User}: JohnMilburk, \text{Role}: junior Researcher)\}$. Then the need for this credential is return back to the user.

On the next interaction step, because the user is a senior researcher, the same declines to present the requested credential as just returning the same query with no presented credentials ($\mathcal{C}_r = \emptyset$). So, the algorithm updates the user's session profile marking the requested credential credential $(\text{User}: JohnMilburk, \text{Role}: junior Researcher)$ as declined. The difference comes when the algorithm re-computes the disclosable credentials as all disclosable credentials from the last interaction minus the newly declined one:

$$\{\text{credential}\,(\text{User}: JohnMilburk, \text{Role}: senior Researcher),$$
$$\text{credential}\,(\text{User}: JohnMilburk, \text{Role}: boardOf Directors)\}$$

Abduction computation returns, as a missing set, the need for credential

$$\text{credential}\,(\text{User}: JohnMilburk, \text{Role}: senior Researcher).$$

Then, because John Milburk is a senior researcher, he presents the just required certificate back to the system and gets the requested service. ∎

Remark 2: Using declined credentials is essential to avoid loops in the process and to guarantee the success of interaction in presence of disjunctive information.

For example suppose we have alternatives in the partner's policy (e.g., "present either a VISA or a Mastercard or an American Express card"). An arbitrary alternative can be selected by the abduction algorithm and on the next interaction step (if the client has declined the credential) the abduction algorithm is informed that the previous solution was not accepted. The process can continue until all credentials have been declined (and access is denied) or a solution is found (and access is granted).

This is all we need for business processes made up by *stateless web services*, in which all decisions are taken on the basis of the current input set of credentials, and which envisaged to be the large majority. This type of decision is characteristic of most logical approaches to access control [24], [4], [5]: we only look at the policy, the request and the set of credentials.

It is possible to extend the approach to stateful autonomic nodes [20] and to a mutual process of trust negotiation [22].

VIII. CORRECTNESS AND COMPLETENESS

At first we introduce some preliminary definitions.

Definition 8.1 (Completeness): If a client has a solution for a request r then he always gets *grant* r.

Definition 8.2 (Soundness): If a client gets *grant* r then he has a solution for r.

Definition 8.3 (Solution Set for a Resource r): Let $\mathcal{P}_\mathcal{A}$ is an access policy and r be a request. A set of credentials $\mathcal{C}_\mathcal{S}$ is a solution set for r according to $\mathcal{P}_\mathcal{A}$ if r is a security consequence of $\mathcal{P}_\mathcal{A}$ and $\mathcal{C}_\mathcal{S}$ ($\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{S} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{S} \not\models \perp$).

Definition 8.4 (Disclosable and Hidden Credentials): Let $\mathcal{P}_\mathcal{D}$ be a disclosure policy, credential c is *disclosable* if there is a set of credentials $\mathcal{C}$ s.t. $c \notin \mathcal{C}$ and $\mathcal{C}$ together with the disclosure policy $\mathcal{P}_\mathcal{D}$ entails c, namely $\mathcal{P}_\mathcal{D} \cup \mathcal{C} \models c$.

A credential c is *hidden* if it is not disclosable.

The intuition behind hidden credentials is that the system *does not ask* for them but *expects* them from the client. So, the information for hidden credentials is obtained by out-of-band sources. Also, from the point of view of a client, hidden credentials are just credentials that someone has told him to provide them when requests a specific service. Hidden credentials are used either to unlock more credentials needed to grant access or used directly to unlock a resource or used for both. So, a client must provide them when initially requests a service. Essentially, the client

can work in *pull* mode with disclosable credentials and must work in *push* mode with hidden credentials.

Definition 8.5 (Hidden Credentials for a Resource r): Hidden credentials for a resource r is the set $\{\mathcal{C}_{\mathcal{H}1},\ldots,\mathcal{C}_{\mathcal{H}n}\}$ where:

1) $\mathcal{C}_{\mathcal{S}i}$, $i = [1..n]$, are all possible solution sets for r,
2) $\mathcal{C}_{\mathcal{H}i} \subseteq \mathcal{C}_{\mathcal{S}i}$, $i = [1..n]$, is the set of all hidden credentials for $\mathcal{C}_{\mathcal{S}i}$.

In particular all solution sets for a resource r could be hidden, i.e. $\mathcal{C}_{\mathcal{H}i} = \mathcal{C}_{\mathcal{S}i}, i = [1..n]$, and we fall back in the standard, classical, access control framework of having only grant/deny decisions. On the other hand, every solution $\mathcal{C}_{\mathcal{S}}$ for r with hidden credentials equal to an empty set is just a solution for r, or can be interpreted as any solution is a solution with hidden credentials where hidden credentials might be equal to empty set. Also it might be a case that some solution sets for r have the same sets of hidden credentials.

Definition 8.6 (Fair Access): Let $\mathcal{P}_{\mathcal{A}}$ be an access control policy and let $\mathcal{C}_{\mathcal{P}_{\mathcal{A}}}$ be the set of ground instances of all credentials occurring in $\mathcal{P}_{\mathcal{A}}$. The policy $\mathcal{P}_{\mathcal{A}}$ *guarantees fair access* if for any request r there exists a set $\mathcal{C}_{\mathcal{S}} \subseteq \mathcal{C}_{\mathcal{P}_{\mathcal{A}}}$ that is a solution for r.

We can check fair access property by simply running the interactive algorithm for all resources $r \in \mathcal{P}_{\mathcal{A}}$ and set up the disclosable credentials to all credentials occuring in $\mathcal{P}_{\mathcal{A}}$. Then for each request r the algorithm should return ask($\mathcal{C}_{\mathcal{M}}$). If for some r it returns *deny* then the fair access property fails.

Definition 8.7 (Fair Interaction): Let $\mathcal{P}_{\mathcal{A}}$ and $\mathcal{P}_{\mathcal{D}}$ be, respectively, an access and disclosure control policies. The policies *guarantee fair interaction* if

1) $\mathcal{P}_{\mathcal{A}}$ guarantees fair access and
2) if $\mathcal{C}_{\mathcal{S}}$ is a solution for a request r and $\mathcal{C}_{\mathcal{H}}$ is the set of hidden credentials for $\mathcal{C}_{\mathcal{S}}$ holds that the visible part of $\mathcal{C}_{\mathcal{S}}$ is disclosable by $\mathcal{P}_{\mathcal{D}} \cup \mathcal{C}_{\mathcal{H}}$, i.e. $\forall c \in (\mathcal{C}_{\mathcal{S}} \setminus \mathcal{C}_{\mathcal{H}}), \ \mathcal{P}_{\mathcal{D}} \cup \mathcal{C}_{\mathcal{H}} \models c$.

The intuition of the fair interaction is that the hidden credentials in a solution set are all that is needed to obtain the disclosure of the remaining disclosable credentials. For example setting up an e-mail account my_name@google.com is an example of fair interaction with hidden credentials. One needs a form that is not available on the site but after the form, duly filled, has been sent some additinal information (credentials) is asked and the account is granted.

The intuition behind unfair interaction policies is that, even if we have all credentials necessary to access, even if we know that some credentials (the hidden ones) must be sent in push mode, yet

this will not be enough to get an answer from the server. We will need to push other credentials that are not needed for access but just to disclose the information on missing credentials.

Assuming that for each request $r \in \mathcal{P}_{\mathcal{A}}$ a service provider knows a priori the set of all hidden credentials $\mathcal{C}_{\mathcal{H}1}, \ldots, \mathcal{C}_{\mathcal{H}n}$ we can check the fair interaction property by running the interactive algorithm n-times with input: r and $\mathcal{C}_{\mathcal{H}i}, \ i = [1..n]$. Then for each request r and each run the algorithm should return either $\texttt{ask}(\mathcal{C}_{\mathcal{M}})$ or *grant*. If for some r in some runs it returns *deny* then the fair interaction property fails.

If a service provider does not know the set of hidden credentials these can again be calculated from the disclosure policies by using the abduction algorithm on the disclosure policy.

Definition 8.8 (Powerful Client): A *powerful client* is a client that whenever receives $\texttt{ask}(\mathcal{C}_{\mathcal{M}})$ returns $\mathcal{C}_{\mathcal{M}}$.

Definition 8.9 (Cooperative Client): A client with a set of credentials $\mathcal{C}$ is a *cooperative client* if whenever receives $\texttt{ask}(\mathcal{C}_{\mathcal{M}})$ returns $\mathcal{C}_{\mathcal{M}} \cap \mathcal{C}$.

Definition 8.10 (Client with Hidden Credentials for a Resource r): A *client with hidden credentials for a resource r* is any client that has the set of hidden credentials $\mathcal{C}_{\mathcal{H}}$ of a solution set for r and whenever requets r it sends $\mathcal{C}_{\mathcal{H}}$ initially.

Definition 8.11 (Monotonic and Non-monotonic Policy): A policy P is *monotonic* if whenever a set of statements $\mathcal{C}$ is a solution set for r according to P ($P \cup \mathcal{C} \models r$) then any superset $\mathcal{C}' \supset \mathcal{C}$ is also a solution set for r acccording to P ($P \cup \mathcal{C}' \models r$).

In contrast, a *non-monotonic* policy is a logic program in which if $\mathcal{C}$ is a solution for r it may exists $\mathcal{C}' \supset \mathcal{C}$ that is not a solution for r, i.e. $P \cup \mathcal{C}' \not\models r$

Definition 8.12 (Resource r Additive Policy): A policy P is *a resource r additive* if for every two sets of statements $\mathcal{C}$ and $\mathcal{C}'$, where $\mathcal{C} \not\subset \mathcal{C}'$ and $\mathcal{C}' \not\subset \mathcal{C}$, that unlock the resource r according to P then also $\mathcal{C} \cup \mathcal{C}'$ unlocks r according to P.

In other words, if you have two solutions for a service you can "add" them and you will still get the service.

Definition 8.13 (Resource r Subset Consistent Policy): A policy P is a *resource r subset consistent* if for every solution set $\mathcal{C}_{\mathcal{S}}$ for r holds that each $\mathcal{C} \subseteq \mathcal{C}_{\mathcal{S}}$ preserves consistency in P, i.e. $P \cup \mathcal{C} \not\models \perp$.

The intuition behind a subset consistent policy is that inconsistency occurs because of separation of duty. So this type of constraints rule out situation in which a client has too many privileges.

Situations where having less credentials than enough to get a service makes the system inconsistent are neither practical nor intuitive from an access logic point of view.

Proposition 8.1 (Sufficient Condition for Subset Consistency): Let $\mathcal{P}_\mathcal{A}$ be an access policy. If the number of negations from a literal in the body of a constraint to a credential in $\mathcal{P}_\mathcal{A}$ is even then $\mathcal{P}_\mathcal{A}$ is *subset consistent*.

Proof: Let assume that the number of negations from a literal in the body of a rule to a credential in $\mathcal{P}_\mathcal{A}$ is even and the policy $\mathcal{P}_\mathcal{A}$ is not subset consistent. Then let $\mathcal{C}_\mathcal{S}$ be a solution set for a request r and let $\mathcal{C} \subseteq \mathcal{C}_\mathcal{S}$ such that $\mathcal{P}_\mathcal{A} \cup \mathcal{C} \models \bot$. Then because $\mathcal{C}_\mathcal{S}$ preserves consistency and the fact that no credential occures in a head of a rule in $\mathcal{P}_\mathcal{A}$ (rf. Def. 6.1) follows that inconsistency occures from reducing the number of credentials in the system. Reducing credentials either directly affects a constraint having negation of a literal that is a credential belonging to $\mathcal{C}_\mathcal{S} \setminus \mathcal{C}$ or indirectly affects a constraint that has a positive literal, which by its side is deduced from a rule in $\mathcal{P}_\mathcal{A}$ that has either negation or a positive literal in the body. If negation in the latter case then it is a missing credential. In both cases the number of negations is 1 which is odd. If we apply recursively the same rule for a positive literal in the latter case we will compute again odd number of negations. In any way we have a contradiction with what we assumed. With this we finish the proof. ∎

Definition 8.14 (Well-behaved Policy): A policy P is *well-behaved* if for all resources $r \in P$

(i) P is *resource r additive* and

(ii) P is *resource r subset consistent*.

The set of well-behaved policies resides between monotonic and non-monotonic policies.

Proposition 8.2: All monotonic policies are well-behaved but the converse is not true.

In one direction the property is immediate: if a policy $\mathcal{P}_\mathcal{A}$ is monotonic then, according Def. 8.11, if a set $\mathcal{C}_\mathcal{S}$ is a solution for a request r then also any superset is. So if $\mathcal{C}_\mathcal{S}'$ is as another solution set for r, so also $\mathcal{C}_\mathcal{S} \cup \mathcal{C}_\mathcal{S}'$ is a solution because of the superset property.

For the other way round we show a counter-example:

$$r_1 \leftarrow \mathcal{C}_A.$$
$$r_1 \leftarrow \mathcal{C}_B.$$
$$r_2 \leftarrow \mathcal{C}_C.$$
$$\leftarrow \mathcal{C}_A, \mathcal{C}_B, \mathcal{C}_C.$$

In our case having $\{\mathcal{C}_A, \mathcal{C}_B, \mathcal{C}_C\}$ bans the client to get either service, which clearly shows that the example is a non-monotonic policy. At the same time, for each of the services we have additive and subset consistent properties so that the policy is well-behaved.

Remark 3: Hereinafter all access policies $\mathcal{P}_\mathcal{A}$ will be *well-behaved* policies and all disclosure policies $\mathcal{P}_\mathcal{D}$ will be *monotonic* policy unless explicitily specified otherwise.

Also we assume that a client initiates a service request with an empty set of presented credentials or, if hidden credentials needed, the presented credentials are the hidden ones. The assumption is important to avoid initial inconsistency and to assure that a client has a successful first step.

Whenever hidden credentials are not explicitly linked in the theorems we assume that for each request r in $\mathcal{P}_\mathcal{A}$ its set of hidden credentials $\mathcal{C}_\mathcal{H} = \emptyset$. The assumption mainly reflects the use of Definition 8.7 for fair interaction. We apply the same definition but with no $\mathcal{C}_\mathcal{H}$ meaning that for any request r each of its solution sets is disclosable by the disclosure policy.

A. Correctness and Completeness

Theorem 8.1 (Soundness): Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy and r a request. If a client gets grant r then he has a solution set $\mathcal{C}_\mathcal{S}$ that unlocks r according to $\mathcal{P}_\mathcal{A}$. This proof is rather straightforward. Let suppose that the client, requested service r, got *grant*. The only way to get it is when $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{P} \models r$. There are two cases: either $\mathcal{C}_\mathcal{P} = \emptyset$ or $\mathcal{C}_\mathcal{P} \neq \emptyset$.

If $\mathcal{C}_\mathcal{P} = \emptyset$ then the resource r is not protected by $\mathcal{P}_\mathcal{A}$, i.e. $\mathcal{P}_\mathcal{A} \models r$. So, the $\emptyset$ is a solution for r and the client has it.

If $\mathcal{C}_\mathcal{P} \neq \emptyset$ then the only way to introduce a credential in $\mathcal{C}_\mathcal{P}$ is by step 2 of the algorithm. Since initially $\mathcal{C}_\mathcal{P} = \emptyset$ so the client has sent a sequence of sets of credentials $\mathcal{C}_{r1}, \ldots, \mathcal{C}_{rn}$ such that $\bigcup_{i=1}^n \mathcal{C}_{ri} = \mathcal{C}_\mathcal{P}$. Then the client has a set of credentials that unlocks it.

Theorem 8.2 (Termination): Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy and r a request. The access control algorithm always terminates.

To prove this claim simply observe that at each interaction the union of the presented credentials or the declined credentials occuring in the access policy always increases. Since this set is bounded by the credentials occuring in the access policy there is always a stage in which either a grant is given (enough presented credentials to unlock the service) or a deny is sent (too many declined credentials to find another set of missing credentials).

B. Completeness

The most important thing is also the most difficult to prove: a client who has the right set of credentials and who is willing to send them to the server, will not be left stranded in our autonomic network and will eventually get a grant.

We prove this result in stages: first for powerful clients and then for cooperative clients. We notice that it is fairly difficult to prove any results for non-cooperative clients: if they are unwilling to send the credentials to the server, how can ever the server grants them access? However, if at least the client is willing to give his credentials if the server guess the right combination can also be captured.

Theorem 8.3 (Completeness for a Powerful Client): Let $\mathcal{P}_A$ be a *non-monotonic* access policy, $\mathcal{P}_D$ be a disclosure policy and r a request. If $\mathcal{P}_A$ and $\mathcal{P}_D$ guarantee fair access and interaction then a powerful client always gets grant r.

Proof: A powerful client requests r with an initial set of presented credentials equal to empty set. Then the algorithm runs steps 1-3. If step 4 succeeds (no credentials are needed for r) then the algorithm returns grant at step 5 and we are done.

If step 4 does not succeed then the algorithm goes to step 5a. At this point $\mathcal{C}_N = \mathcal{C}_P = \emptyset$. In this case $\mathcal{C}_D$ consists of all credentials disclosable by $\mathcal{P}_D$. Then in step 5b the abduction algorithm will return a set $\mathcal{C}_M$ that unlocks r because

 (i) $\mathcal{P}_A$ guarantees fair access and so a solution set $\mathcal{C}_S$ for r exists,

 (ii) since $\mathcal{P}_D$ satisifies the property fair interaction and $\mathcal{C}_H = \emptyset$ so $\mathcal{C}_S$ is disclosed by $\mathcal{P}_D$,

(iii) clearly $\mathcal{C}_S$ is a subset of $\mathcal{C}_D$, because $\mathcal{C}_D$ consists of all credentials disclosable by $\mathcal{P}_D$.

So, step 5c is not reached and in step 5d the algorithm returns `ask(`$\mathcal{C}_M$`)` which satisfies the two conditions of step 5b.

If there is only one solution that unlocks r then $\mathcal{C}_M = \mathcal{C}_S$.

Since the client is a powerful client then on the next interaction he returns $\mathcal{C}_M$. Then the algorithm updates $\mathcal{C}_P = \mathcal{C}_M$ and $\mathcal{C}_N = \emptyset$ and because $\mathcal{C}_M$ satisfies the two conditions in step 5b, in the last interaction, so it also satisfies the conditions in step 4 and in step 5 it returns *grant*. ■

Theorem 8.4 (Completeness for a Cooperative Client): Let $\mathcal{P}_A$ be an access policy, $\mathcal{P}_D$ be a disclosure policy and r a request. If $\mathcal{P}_A$ and $\mathcal{P}_D$ guarantee fair access and interaction then if

a cooperative client has a set of credentials $\mathcal{C}_\mathcal{S}$ that unlocks r according to $\mathcal{P}_\mathcal{A}$ then the client always gets grant r.

Proof: We will proof it in two steps. First step, by induction, showing that in a single interaction if a cooperative client does not get grant r then he gets $\texttt{ask}(\mathcal{C}_\mathcal{M})$. In other words, he will never receive denial by the algorithm. Second step, we will show that if the first step is true then a cooperative client with a solution set $\mathcal{C}_\mathcal{S}$ always gets grant r.

Step 1.

Proof by induction on the interaction steps:

> **Inter. 1**: Client requests service r together with an initial set of credentials $\mathcal{C}_r = \emptyset$ and we fall back exactly in the proof of Theorem 8.3 for that interaction step.
>
> **Inter. N**: Here we use the induction hypothesis that the client fails to get grant r and gets $\texttt{ask}(\mathcal{C}_\mathcal{M})$.
>
> **Inter. N+1**: Here, let suppose that the client fails to get grant r in step 4. The only way it fails is that there is no solution set in $\mathcal{C}_\mathcal{P}$. It is because in $\mathcal{C}_\mathcal{P}$ there are only credentials partially compiled from solutions for r, i.e. $\mathcal{C}_\mathcal{P} \subset (\mathcal{C}_{\mathcal{M}1} \cup \ldots \cup \mathcal{C}_{\mathcal{M}n})$. Then since $\mathcal{P}_\mathcal{A}$ is *well-behaved* so $\mathcal{C}_\mathcal{P}$ preserves consistency in $\mathcal{P}_\mathcal{A}$.
>
> So, after the check failed in step 4 the algorithm computes the set of disclosable credentials $\mathcal{C}_\mathcal{D}$ as all credentials disclosable by $\mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{P}$ minus all already declined and presented credentials.
>
> Because $\mathcal{P}_\mathcal{A}$ and $\mathcal{P}_\mathcal{D}$ guarantee fair access and interaction so $\mathcal{C}_\mathcal{S} \subseteq (\mathcal{C}_\mathcal{D} \cup \mathcal{C}_\mathcal{P})$ but $\mathcal{C}_\mathcal{S} \nsubseteq \mathcal{C}_\mathcal{P}$. Then at least the set difference $\mathcal{C}_\mathcal{S} \setminus \mathcal{C}_\mathcal{P}$ will be computed by the abduction engine because: (i) $(\mathcal{C}_\mathcal{S} \setminus \mathcal{C}_\mathcal{P}) \subseteq \mathcal{C}_\mathcal{D}$ and (ii) $(\mathcal{C}_\mathcal{S} \cup \mathcal{C}_\mathcal{P}) \subset (\mathcal{C}_{\mathcal{M}1} \cup \ldots \cup \mathcal{C}_{\mathcal{M}n} \cup \mathcal{C}_\mathcal{S})$ preserves consistency (rf. Def. 8.14 and Def. 8.13). And so the step 5c is skipped the client gets $\texttt{ask}(\mathcal{C}_\mathcal{M})$.

Step 2. So in Step 1 we proved that if a cooperative client does not get grant r, in a single interaction step, he gets $\texttt{ask}(\mathcal{C}_\mathcal{M})$. Then we have to prove that in a finite number of steps a cooperative client will always get grant r.

There is a finite number of solutions for each request r simply because $\mathcal{P}_\mathcal{A}$ consists of a finite number of statements. The abduction reasoning service at each interaction computes different solution wrt the solutions computed in previous interactions because from the disclosable credentials we remove all those credentials from the previous interactions (rf. step 5a in Fig. 4).

So, since there are finite solution sets for r and since the client has one of them therefore

according to what we proved in **Step 1** the client in a finite number of interaction steps will disclose $C_\mathcal{S}$, i.e. $C_\mathcal{S} \subseteq C_\mathcal{P}$, and get grant r. ∎

Theorem 8.5 (Completeness for a Powerful Client with Hidden Credentials): Let $\mathcal{P_A}$ be an access policy, $\mathcal{P_D}$ be a disclosure policy and r a request. If $\mathcal{P_A}$ and $\mathcal{P_D}$ guarantee fair access and interaction then a powerful client with hidden credentials $C_\mathcal{H}$ for r always gets grant r.

Proof: A powerful client requests r with an initial set of presented credentials equal to the set of hidden credentials, i.e. $C_r = C_\mathcal{H}$. Then the algorithm runs steps 1-3. At this point $C_\mathcal{P} = C_\mathcal{H}$ and $C_\mathcal{N} = \emptyset$. If step 4 succeeds, $C_\mathcal{H}$ itself is a solution set for r, then the algorithm returns grant at step 5 and we are done.

If step 4 does not succeed then the algorithm goes to step 5a. In this case $C_\mathcal{D}$ consists of all credentials disclosable by $\mathcal{P_D}$ and $C_\mathcal{H}$ because $C_\mathcal{P} = C_\mathcal{H}$. Then in step 5b the abduction algorithm will return a set $C_\mathcal{M}$ because at least one such set exists:

(i) $\mathcal{P_A}$ and $\mathcal{P_D}$ guarantee fair access and interaction and so for the solution set $C_\mathcal{S}$, corresponding to the set of hidden credentials $C_\mathcal{H}$ that the client has, its visible part $C_\mathcal{S} \backslash C_\mathcal{H}$ will be disclosed by $\mathcal{P_D}$ and $C_\mathcal{H}$ (rf. Def. 8.7),

(ii) clearly $(C_\mathcal{S} \backslash C_\mathcal{H}) \subseteq C_\mathcal{D}$, because $C_\mathcal{D}$ consists of all credentials disclosable by $\mathcal{P_D} \cup C_\mathcal{H}$,

(iii) according to Definition 8.5 the set $C_\mathcal{H} \subseteq C_\mathcal{S}$ and so the set $C_\mathcal{S} \backslash C_\mathcal{H}$ together with $C_\mathcal{P}$ preserve consistency in $\mathcal{P_A}$ (simply because $(C_\mathcal{S} \backslash C_\mathcal{H}) \cup C_\mathcal{P} = C_\mathcal{S}$ is a solution set),

So, step 5c is not reached and in step 5d the algorithm returns `ask(`$C_\mathcal{M}$`)` which satisfies the two conditions of step 5b.

It may be the case in which the set $C_\mathcal{H}$ unlocks more than one solution sets if those soluiton sets have the same set of hidden credentials.

Since the client is a powerful client then on the next interaction he returns $C_\mathcal{M}$. Then the algorithm updates $C_\mathcal{P} = C_\mathcal{H} \cup C_\mathcal{M}$ and $C_\mathcal{N} = \emptyset$ and because $C_\mathcal{M}$ satisfies the two conditions in step 5b, from the last interaction, so it also satisfies the conditions in step 4 and in step 5 it returns *grant*. With this we finish the proof. ∎

Theorem 8.6 (Completeness for a Cooperative Client with Hidden Credentials): Let $\mathcal{P_A}$ be an access policy, $\mathcal{P_D}$ be a disclosure policy and r a request. If $\mathcal{P_A}$ and $\mathcal{P_D}$ guarantee fair access and interaction then if a cooperative client with hidden credentials $C_\mathcal{H}$ for r has a solution set $C_\mathcal{S}$ for $C_\mathcal{H}$ then the client always gets grant r.

Proof: Analogously of theorem 8.4 we will proof it in two steps. First step, by induction,

showing that in a single interaction if the client does not get grant r then he gets $\mathtt{ask}(\mathcal{C}_\mathcal{M})$. In other words, he will never receive denial by the algorithm. Second step, we will show that if the first step is true then the client with a solution set $\mathcal{C}_\mathcal{S}$ always gets grant r.

Proof by induction on the interaction steps:

Inter. 1: Client requests service r together with an initial set of credentials $\mathcal{C}_r = \mathcal{C}_\mathcal{H}$ and we fall back exactly in the proof of Theorem 8.5 for that interaction step.

Inter. N: Here we use the induction hypothesis that the client fails to get grant r and gets $\mathtt{ask}(\mathcal{C}_\mathcal{M})$.

Inter. N+1: Now, let suppose that the client fails to get grant r in step 4. Then the only difference with the respective interaction step in the proof of Theorem 8.4 is that the client's solution set $\mathcal{C}_\mathcal{S}$ is still disclosable by $\mathcal{P}_\mathcal{D}$ and $\mathcal{C}_\mathcal{P}$ because $\mathcal{C}_\mathcal{H}$ is already in the set $\mathcal{C}_\mathcal{P}$, see Definition 8.5. So, according to fair interaction property follows that $\mathcal{C}_\mathcal{S} \setminus \mathcal{C}_\mathcal{H}$ is visible and the abduction service at least finds it as a solution.

The rest of the proof can be done along the same line as in Theorem 8.4. ∎

IX. Implementation

We have actually implemented the system for access control for abduction and deduction using protocols over web services and a front-end to a state-of-the-art abduction/deduction engine and integrating it with a system for PMI (privilege management infrastructure).

For the implementation of the framework we have chosen Collaxa[3] manager. Collaxa server supports many standards as BPEL4WS, WSDL, SOAP, etc. and interoperates with platforms as BEA's WebLogic and Microsoft .NET. So, this makes it well-suited for the purposes of the framework. The main idea of the work is that using BPEL4WS specification [30] we can orchestrate the requirements and communications between client and partners in an automatic and transparent way via a main authorization server.

A preliminary prototype of the system, especially the authorization server, has been done under Collaxa. We have also defined all WDSL interfaces and XML schemas for the messages that entities in the framework exchange with each other.

---

[3]www.collaxa.com | www.oracle.com/technology/bpel

For the actual crypto infrastructure we decided to use PERMIS[4] [6]. We chose PERMIS because it implements RBAC using entirely X.509 Identity and Attribute Certificates [15]. It allows for creating, verifying and validating attribute certificates and for storing and allocating them using LDAP directories [40]. For the integration with PERMIS, we extend the PERMIS's Access Decision Function (ADF) with the functionality of our model such that PERMIS validates and gathers client's credentials on its own and then asks our algorithm for an access decision (next possible step) presenting the newly collected credentials.

For the implementation of the algorithm, presented in the paper, we use DLV[5] (a disjunctive datalog system with negations and constraints) as a core engine. We have done a wrapper to the DLV system that manages all internal computations, queries and transformations to and from DLV. Additional details on the implementation of the logical steps can be found in [21].

X. RELATED WORK

As we mentioned in the introduction, access control for autonomic communication borrows some aspect of trust management and some aspects of workflow security. Among these models we find a number of relevant works: for workflows[4], web services [33], role based access control on the web [13], [34], tasks [17] and DRM [33], possibly coupled by sophisticated policy combination algorithms. However, they have mostly remained within the classical framework – all decisions of grant/deny are based on checking that request would follow from the policy and the presented set of credentials.

The works on trust negotiations[35], [44] focus on communication and infrastructure and assume that requests and counter requests have been somehow calculated from the access policy. Also the formal models on credential-based access control and policy combination [4], [24], [16], [42] dont treat the problem of inferring missing credentials from failed requests, as they are within the same frame of mind of inferring successful requests from present credentials. Also standardization efforts like the XACML proposals [14] gives rules for deriving what is right (evaluating policies) and not rule for understanding what went wrong.

Also a recent proposal by Bonatti and Samarati [5] that has the explicit focus on access and release control is not fully on target. In a nutshell, the request is received, the policy rules are

---

[4]www.permis.org

[5]www.dlvsystem.com

filtered for relevance, the relevant rules are partially evaluated and sent to the client. The client will have to figure out which are the credentials (this is not discussed in the paper) and then will evaluate these credentials according its release policy.

The first problem is that demanding clients to analyse security policies is not acceptable here. The second problem is that after a suitable number of queries the entire policy of the server would be disclosed to the client or to the server orchestrating the process. Here we only disclose the needed credentials and not the rules of the policy whose structure remains hidden to the client. Furthermore, the relevancy filtering approach only works for flat (monotone) policies, in which for every request we list all its credentials.

The other key proposal on trust negotiation by Yu et al. [44], offers a dual view w.r.t Bonatti and Samarati[5]. Loosely speaking, each credential is associated to a policy (a boolean expression) denoting the credentials that a client must have already provided for its safe disclosure. By a step wise process the parties can exchange credentials or policy rules until the desired resource is released. The paper provides safe sequences of disclosure in a rather ad-hoc fashion building upon trees rather than logical formalization. As a consequence they can only treat monotone policies and it is not possible to define notions of consistency of policies and disclosure of policies in presence of constraints (e.g. separation of duty). Another limitation of the paper is that it interlocks the access and the release policy into one. So, as the authors acknowledge [44, page 21], it is impossible to access resources if some of the needed credentials cannot be disclosed at some point. Furthermore, the need for intermediate credential disclosure calls for a structuring of policy rules that may be counter-intuitive from the point of view of access control. For the disclosure process to take place such natural composition is not possible when using Yu et al. framework [44].

From the literate of policy-based network management (or self-management), we have a number of works that also follow within the same mindset: given some fact and our policy we derive the actions. For example, the IETF Policy working group is defining a framework for QoS based on the X.500 directory [39]. The IETF "policies" have the form if "set of conditions" then do "a set of actions". These rules are also fairly limited as they do not support rules that can be dynamically activated by events to reconfigure the system according to changed circumstances. Industry solutions [29] are hardly better than the IETF proposal and sometimes turns out just to be a graphical interface over DiffServ network policies. Other accademic proposals for DiffServ

such as that by Keller et al. [19] may allow for more sophisticated policies and contractual monitoring of those policies. Other proposals make use of policies including obligations [37], [27] but still remains in the setting if-then.

The most intriguing proposal is the bacterial algorithm presented by Marshal et al. for network management [28]. Loosely speaking, the algorithm is constituted by autonomous (but centrally programmed) controllers which autonomously replicate policies that improve its performance and de-activate policies that degrade performance (policies are evaluated using a cost-revenues filter). In this way, good policies should proliferate and poor policies should die out. This proposal is particularly interesting because it suggest using another reasoning service: *induction* [31]. This work can be the subject of future investigations.

XI. CONCLUSIONS

In this paper we proposed a framework for policy-based self-managed access control in autonomic communication. The framework is grounded in a formal model based on Datalog with the stable model semantics. The key idea is that in an autonomic network a client may have the right credentials but may not know them and thus a autonomic communication server needs a way to avoid leaving the client stranded.

We have proposed a solution to this problem by extending classical policy-based access control models with an advance reasoning service: abduction. Building on top of this service we have presented an interactive access control algorithm that computes on the fly the missing credentials needed for a client to get access.

We also enriched the framework over existing policy-based approach to access control by introducing the difference between *disclosable* and *hidden* credentials and between *monotonic* and *well-behaved* policies. The first distinction addresses the behaviour of an autonomic node allowing it to dynamically protect the privacy of his policies by specifying which credentials are hidden and which are not. This allows the server to restrict access to selected clients. The latter distinction extends our work on a wider set of policy languages wrt already existing approaches [5], [44], [25].

We have also sketched a the implementation of the systems using web-services.

Future work is in the direction of characterizing the complexity of the framweork[6], extending it to cope with mutual negotiation (a preliminary result is presented in [22]), and fully integrate our implementation with a privilege management infrastructure.

REFERENCES

[1] APT, K. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier, 1990.

[2] ATLURI, V., CHUN, S. A., AND MAZZOLENI, P. A Chinese wall security model for decentralized workflow systems. In *Proceedings of the 8th ACM conference on Computer and Communications Security* (2001), ACM Press, pp. 48–57.

[3] BERTINO, E., CATANIA, B., FERRARI, E., AND PERLASCA, P. A logical framework for reasoning about access control models. In *Proceedings of the Sixth ACM Symposium on Access control models and technologies* (2001), ACM Press, pp. 41–52.

[4] BERTINO, E., FERRARI, E., AND ATLURI, V. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security (TISSEC) 2*, 1 (1999), 65–104.

[5] BONATTI, P., AND SAMARATI, P. A unified framework for regulating access and information release on the web. *Journal of Computer Security 10*, 3 (2002), 241–272.

[6] CHADWICK, D. W., AND OTENKO, A. The PERMIS X.509 role-based privilege management infrastructure. In *Seventh ACM Symposium on Access Control Models and Technologies* (2002), ACM Press, pp. 135–140.

[7] CLARKE, D., ELIEN, J.-E., ELLISON, C., FREDETTE, M., MORCOS, A., AND RIVEST, R. L. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security 9*, 4 (2001), 285–322.

[8] DAMIANOU, N., DULAY, N., LUPU, E., AND SLOMAN, M. The Ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY)* (January 2001), Springer-Verlag, pp. 18–38.

[9] DAS, S. *Deductive Databases and Logic Programming*. Addison-Wesley, Reading, MA, 1992.

[10] ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B. M., AND YLONEN, T. *SPKI Certificate Theory*, September 1999. IETF RFC 2693.

[11] GELFOND, M., AND LIFSCHITZ, V. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming (ICLP'88)* (1988), R. Kowalski and K. Bowen, Eds., MIT-Press, pp. 1070–1080.

[12] GEORGAKOPOULOS, D., HORNICK, M. F., AND SHETH, A. P. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases 3*, 2 (April 1995), 119–153.

[13] GIURI, L. Role-based access control on the web. *ACM Transactions on Information and System Security (TISSEC) 4*, 1 (2001), 37–71.

[14] GODIK, S., AND MOSES, T. *eXtensible Access Control Markup Language (XACML)*. OASIS, February 2003. www.oasis-open.org/committees/xacml/.

---

[6]General abduction lay at the second level of the polynomial hierarchy but our problems are at the same time more specialized (e.g. credentials are occurring only positively in the rules) and more general (we have hierarchies of roles so subset or cardinality minimality does not really apply).

[15] ITU-T RECOMMENDATION X.509:2000(E) | ISO/IEC 9594-8:2001(E). The directory: Public-key and attribute certificate frameworks.

[16] JAJODIA, S., SAMARATI, P., SUBRAHMANIAN, V. S., AND BERTINO, E. A unified framework for enforcing multiple access control policies. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data* (1997), ACM Press, pp. 474–485.

[17] JOSHI, J. B. D., AREF, W. G., GHAFOOR, A., AND SPAFFORD, E. H. Security models for web-based applications. *Communications of the ACM 44*, 2 (2001), 38–44.

[18] KANG, M. H., PARK, J. S., AND FROSCHER, J. N. Access control mechanisms for inter-organizational workflow. In *Proceedings of the Sixth ACM Symposium on Access control models and technologies* (2001), ACM Press, pp. 66–74.

[19] KELLER, A., KAR, G., LUDWIG, H., DAN, A., AND HELLERSTEIN, J. Managing dynamic services: A contract-based approach to a conceptual architecture. In *Eighth Network Operations and Management Symposium* (April 2002).

[20] KOSHUTANSKI, H., AND MASSACCI, F. E pluribus unum: Deduction, abduction and induction, the reasoning services for access control in autonomic communication. In *Proceedings of the 1st IFIP TC6 WG6.6 International Workshop on Autonomic Communication (WAC)* (Berlin, Germany, October 2004). to appear.

[21] KOSHUTANSKI, H., AND MASSACCI, F. Interactive access control for Web Services. In *Proceedings of the 19th IFIP International Information Security Conference (SEC 2004)* (Toulouse, France, August 2004), Kluwer Press, pp. 151–166.

[22] KOSHUTANSKI, H., AND MASSACCI, F. An interactive trust management and negotiation scheme. In *Proceedings of the 2nd International Workshop on Formal Aspects of Security and Trust (FAST)* (Toulouse, France, August 2004), Kluwer Press, pp. 139–152.

[23] LEONE, N., PFEIFER, G., AND ET AL. The DLV system. In *the 8th European Conference on Artificial Intelligence (JELIA)* (September 2002), vol. 2424 of *Lecture Notes in Computer Science*, Springer, pp. 537–540.

[24] LI, N., GROSOF, B. N., AND FEIGENBAUM, J. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC) 6*, 1 (2003), 128–171.

[25] LI, N., AND MITCHELL, J. C. RT: A role-based trust-management framework. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX III)* (Los Alamitos, California, April 2003), IEEE press, pp. 201–212.

[26] LI, N., WINSBOROUGH, W. H., AND MITCHELL, J. C. Distributed credential chain discovery in trust management. *Journal of Computer Security 11*, 1 (February 2003), 35–86.

[27] LYMBEROPOULOS, L., LUPU, E., AND SLOMAN, M. An adaptive policy based framework for network services management. *Plenum Press Journal of Network and Systems Management 11*, 3 (September 2003), 277–303.

[28] MARSHALL, I., GHARIB, H., HARDWICKE, H., AND ROADKNIGH, C. A novel architecture for active service management. In *IEEE/IFIP International Symposium on Intergrated Network Management* (May 2001), pp. 795–810.

[29] CISCO COPS QOS POLICY MANAGER PRODUCT DOCUMENTATION. http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/qos/qpm21/index.htm.

[30] CURBERA ET AL, F. *Business Process Execution Language for Web Services (BPEL4WS)*. BEA, IBM, Microsoft, May 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

[31] MUGGLETON, S., AND DE RAEDT, L. Inductive logic programming: Theory and methods. *JLP 19/20* (1994), 629–679.

[32] NIEMELÄ, I., SIMONS, P., AND SOININEN, T. Stable model semantics of weight constraint rules. In *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning* (December 1999), Springer-Verlag.

[33] PARK, J., AND SANDHU, R. Towards usage control models: beyond traditional access control. In *Seventh ACM Symposium on Access Control Models and Technologies* (2002), ACM Press, pp. 57–64.

[34] PARK, J. S., AND SANDHU, R. RBAC on the Web by smart certificates. In *Proceedings of the fourth ACM workshop on Role-based access control* (1999), ACM Press, pp. 1–9.

[35] ROSCHEISEN, M., AND WINOGRAD, T. A communication agreement framework for access/action control. In *Proceedings of the Symposium on Security and Privacy* (1996), IEEE Press, pp. 154–163.

[36] SHANAHAN, M. Prediction is deduction but explanation is abduction. In *Proceedings of IJCAI '89* (1989), Morgan Kaufmann, pp. 1055–1060.

[37] SLOMAN, M., AND LUPU, E. Policy specification for programmable networks. In *Proceedings of the First International Working Conference on Active Networks* (1999), Springer-Verlag, pp. 73–84.

[38] SMIRNOV, M. Rule-based systems security model. In *Proceedings of the Second International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS)* (2003), Springer, pp. 135–146.

[39] SNIR, Y., RAMBERG, Y., STRASSNER, J., AND COHEN, R. Policy framework QoS information model, April 2001. Internet Draft, draft-ietf-policy-qos-info-model-03.txt.

[40] WAHL, M., HOWES, T., AND KILLE, S. Lightweight Directory Access Protocol (v3), December 1997. RFC 2251.

[41] WEEKS, S. Understanding trust management systems. In *IEEE SS&P-2001* (2001), IEEE Press.

[42] WIJESEKERA, D., AND JAJODIA, S. Policy algebras for access control the predicate case. In *Proceedings of the 9th ACM conference on Computer and Communications Security* (2002), ACM Press, pp. 171–180.

[43] WINSBOROUGH, W., SEAMONS, K., AND JONES, V. Automated trust negotiation. In *Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX)* (2000), vol. 1, IEEE Press, pp. 88–102.

[44] YU, T., WINSLETT, M., AND SEAMONS, K. E. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security (TISSEC) 6*, 1 (2003), 1–42.