E Pluribus Unum*

Deduction, Abduction and Induction, the Reasoning Services for Access Control in Autonomic Communication

Hristo Koshutanski and Fabio Massacci

Dip. di Informatica e Telecomunicazioni - Univ. di Trento
via Sommarive 14 - 38050 Povo di Trento (ITALY)
{hristo, massacci}@dit.unitn.it

Abstract. Autonomic Communication is a new paradigm for dynamic network integration. An Autonomic Network crosses organizational boundaries and is provided by entities that see each other just as business partners. Policy-base network anagement 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 partner may guess a priori what credentials will be sent by clients and clients may not know a priori which credentials are required for completing a service requiring the orchestration of many different autonomic nodes. We propose a logical framework and a Web-Service based implementation for reasoning about access control for Autonomic Communication. Our model is based on interaction and exchange of requests for supplying or declining missing credentials. We identify the formal reasoning services that characterise the problem and sketch their implementation.

1 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* (e.g. [1] or the IEEE Policy Workshop series). The intuition is that actions of nodes "controlling" the communication are automatically derived from policies. Nodes look at events and requests presented to them, evaluates the rules of their policies and derive actions [1, 2]. Policies can be "simple" `iptables` rules for Linux firewalls (see `http://www.netfilter.org/`) or complex logical policies expressed in languages such as Ponder [3].

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 them and guarantee interoperation. Nodes are

* This work is partially funded by EU IST-2001-37004 WASP, EU IST E-NEXT NoE, FIRB RBNE0195K5 ASTRO and FIRB RBAU01P5SS projects.

M. Smirnov (Ed.): WAC 2004, LNCS 3457, pp. 179–190, 2005.

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 [4, 5, 6] it takes the credential-based view: access to services is offered by autonomic nodes on their own and the decision to grant or deny access must rely on attribute credentials sent by the client. In contrast with these systems, we have a continuous process and assignment of permissions to credentials must look beyond the single access decision.

From workflow access control [7, 8, 9, 10] we borrow all classical problems such as dynamic assignment of roles to users, separation of duties, and assignment of permissions to users according the least privilege principles. In contrast with such 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.

Astracting away the details of the policy implementation, we can observe that only one reasoning service is actually used by policy based self-management: *deduction*. Given a policy and a set of additional facts and events, we find out all consequences (actions or obligations) of the policy and the facts, i.e. whether granting the request can be deduced from the policy and the current facts. Policies can be different [11, 6, 12, 8] but the kernel reasoning service is the same.

Autonomic communication needs at least another reasoning service: *abduction* [13]. Loosely speaking, abduction is deduction in reverse: given a policy and a request for access to services, find the credentials/events that would grant access, i.e. a (possibly minimal) set of facts that added to the policy would make the request a logical consequence. Abduction is a core service for the *interactive access control* framework in autonomic communication. In this framework a client may be asked on the fly for additional credentials and the same may disclose them or decline to provide them. We need an interactive control on both the client and server sides whenever the client requires some evidence from the server before disclosing his own credentials.

We might also use *induction* [14]: given a heuristic function to measure the goodness of a rule and some examples of granted and denied requests, invent the access policies covering the positive examples and not the negative ones.

Here, we sketch the reasoning framework for access control for autonomic communication based on interaction for supplying missing credentials or for revoking "wrong" credentials (§2). We identify the reasoning services deduction vs abduction (§4), and induction (§7), and sketch the solution for stateful access control (§5) and mutual negotiation (§6). A running example (§3) makes discussion concrete. A discusssion of future challenges concludes the paper.

2 Access Control with Security Policies

Using Datalog and logic programs for security policy is customary in computer security [11, 6, 12, 8] and our formal model is based on normal logic programs under the stable model semantics [15]. We have predicates for requests, creden-

$\mathsf{Role}:R_i \succ \mathsf{Role}:R_j$ when role $\mathsf{Role}:R_i$ dominates role $\mathsf{Role}:R_j$.
$\mathsf{Role}:R_i \succ_{\mathsf{WebServ}:S} \mathsf{Role}:R_j$ when role $\mathsf{Role}:R_i$ dominates, for service $\mathsf{WebServ}:S$, the role $\mathsf{Role}:R_j$.
$\mathsf{assign}\,(P, \mathsf{WebServ}:S)$ when an access to the service $\mathsf{WebServ}:S$ is granted to P. Where P can be either a $\mathsf{Role}:R$ or $\mathsf{User}:U$.

(a) Predicates for assignments to Roles and Services

$\mathsf{declaration}\,(\mathsf{User}:U)$ it is a statement by the $\mathsf{User}:U$ for its identity.
$\mathsf{credential}\,(\mathsf{User}:U, \mathsf{Role}:R)$ when $\mathsf{User}:U$ has a credential activating $\mathsf{Role}:R$.
$\mathsf{credentialTask}\,(\mathsf{User}:U, \mathsf{WebServ}:S)$ when $\mathsf{User}:U$ has the right to access $\mathsf{WebServ}:S$.

(b) Predicates for Credentials

$\mathsf{running}\,(P, \mathsf{WebServ}:S, \mathsf{number}:N)$ when the $\mathsf{number}:N$-th activation of $\mathsf{WebServ}:S$ is executed by P.
$\mathsf{abort}\,(P, \mathsf{WebServ}:S, \mathsf{number}:N)$ if the $\mathsf{number}:N$ activation of $\mathsf{WebServ}:S$ within a workflow aborts.
$\mathsf{success}\,(P, \mathsf{WebServ}:S, \mathsf{number}:N)$ if the $\mathsf{number}:N$-th activation of $\mathsf{WebServ}:S$ within a workflow successfully executes.
$\mathsf{grant}\,(P, \mathsf{WebServ}:S, \mathsf{number}:N)$ if the $\mathsf{number}:N$ request of $\mathsf{WebServ}:S$ has been granted
$\mathsf{deny}\,(P, \mathsf{WebServ}:S, \mathsf{number}:N)$ if the $\mathsf{number}:N$-th request of $\mathsf{WebServ}:S$ has been denied.

(b) Predicates for System's History and State

Fig. 1. Predicates used in the model

tials, assignments of users to roles and of roles to services, see Figure 1. They are self explanatory, except for role dominance: a role dominate another if it has more privileges. We have constants for users identifiers, denoted by $\mathsf{User}:U$, for roles, denoted by $\mathsf{Role}:R$, and one for services, denoted by $\mathsf{WebServ}:S$.

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 former is used for making decision about access to the services offered by the partner. The latter is used to decide the credentials whose need can be potentially disclosed to the client.

We keep a set of *active (unrevoked) credentials* $\mathcal{C}_{\mathcal{P}}$ presented by the client in past requests to other services offered by the same server, and the set of *declined credentials* $\mathcal{C}_{\mathcal{N}}$ compiled from the client's past interactions. To request a service the client submit a set of *presented credentials* $\mathcal{C}_p$, a set of *revoked credentials* $\mathcal{C}_{\mathcal{R}}$ and a *service request* r. We assume that $\mathcal{C}_p$ and $\mathcal{C}_{\mathcal{R}}$ are disjoint. In this context, $\mathcal{C}_{\mathcal{N}}$ is assigned the difference between the missing credentials $\mathcal{C}_{\mathcal{M}}$, the client was asked in the previous interaction, and the ones presented now. For stateful autonomic nodes we'll also need the *history of access to services* $\mathcal{H}$.

3 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, and that there are three main access types to the resources: *read* – access to data residing on the Planet-Lab machines; *run* – access to data and possibility to run processes on the machines; and *configure* – including the previous two types of accesses plus the possibility of configuring network services on the machines.

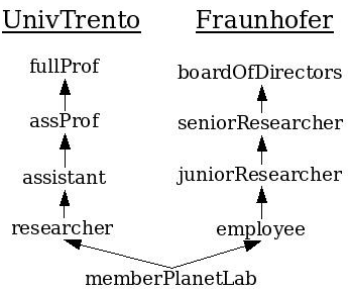

Fig. 2. Joint Hierarchy Model

All Planet-Lab credentials (certificates) are signed and issued by trusted authorities and the crypto validation is performed before the actual access control process. In other words, a preprocessing step validates and transforms the certificates into a form suitable for the formal model – $\mathsf{credential}(\mathsf{User}:U, \mathsf{Role}:R)$.

Fig. 2 shows the role hierarchy, 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. Fig. 3 shows the access and disclosure policies. $\mathsf{authNetwork}(IP, DomainName)$ is domain specific: the first argument is the IP address of the authorized network endpoint (the client's machine) and the second one the domain where the IP address comes from.

Example 1. Rules (1,2) give access to the shared network content to everybody from UniTrento and Fraunhofer, regardless of IP and role. For rules (6,7), if a user has got a disk access and is a researcher at UniTrento or junior researcher at Fraunhofer, it has additional rights. Rules (10,11) give full access from anywhere only to members of the board of directors and to full professors.

Example 2. 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.

Example 3. Rules (1,2) in the disclosure policy show 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; and rule (4) discloses (upgrades) the need of a higher role-position credential.

4 Deduction vs Abduction

The basic reasoning service for policy-based approches is deduction:

Definition 1 (Logical Consequence and Consistency). *We use the symbol $P \models L$, where P is a policy and L is either a credential or a service request, to specify that L is a logical consequence of a policy P. P is* consistent *($P \not\models \bot$) if there is a model for P.*

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 $(User, Role)$, $Role \succeq boardOfDirectors$.

Release 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 the Online Library

This reasoning service is used in most logical formalizations [16]: if the request r is a consequence of the policy and the credentials (i.e. $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_p \models r$), then access is granted otherwise it is denied.

Example 4. A request coming from *dottorati.dit.unitn.it* with IP *193.168.205.11* for access to a fellowship application form on the subnet is granted by rule (3).

The next service is abduction: given a policy and a request, find the credentials that added to the policy would allow to grant the request.

Definition 2 (Abduction). *The abductive solution over a policy P, a set of predicates (credentials) H (with a partial order $\prec$ over subsets of H) and a ground literal L is a set of ground atoms E such that: (i) $E \subseteq H$, (ii) $P \cup E \models L$, (iii) $P \cup E \not\models \bot$, (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.

This reasoning service is used in the overall interactive access control algorithm shown in Fig. 4. Initially the client will send a set of client's credentials $\mathcal{C}_p$and a service request r. Then we update client's profile, i.e. declined and active credentials and check whether the active credentials unlock r according to $\mathcal{P}_{\mathcal{A}}$. In the case of denial, we compute all credentials disclosable from $\mathcal{C}_{\mathcal{P}}$ according to $\mathcal{P}_{\mathcal{D}}$ and from the resulting set remove all $\mathcal{C}_{\mathcal{N}}$. Then we compute all possible subsets of $\mathcal{C}_{\mathcal{D}}$ that are consistent with the access policy $\mathcal{P}_{\mathcal{A}}$ and, at the same

Global vars: $\mathcal{C}_\mathcal{N}$, $\mathcal{C}_\mathcal{P}$;
Internal input: $\mathcal{P}_\mathcal{A}$, $\mathcal{P}_\mathcal{D}$;
Output: grant/deny/ask($\mathcal{C}_\mathcal{M}$);

1. **client's input**: $\mathcal{C}_p$ and r,
2. update $\mathcal{C}_\mathcal{N} = (\mathcal{C}_\mathcal{N} \cup \mathcal{C}_\mathcal{M}) \setminus \mathcal{C}_p$, where $\mathcal{C}_\mathcal{M}$ is from the last interaction,
3. update $\mathcal{C}_\mathcal{P} = \mathcal{C}_\mathcal{P} \cup \mathcal{C}_p$,
4. verify that 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. if the check succeeds then return ***grant*** else
   (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) if no set $\mathcal{C}_\mathcal{M}$ exists then return ***deny*** else
   (d) return ***ask($\mathcal{C}_\mathcal{M}$)*** and iterate.

Fig. 4. Interactive Access Control Algorithm

time, grant r. Out of all these sets (if any) the algorithm selects the minimal one. We point out that the minimality criterion could be different for different contexts (see [17] for some examples).

Remark 1. 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.

Example 5. Assuming the access and release policies in Figure 3, let us play the following 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*. According to the access policy the credentials are not enough to get full access and so the request would be denied.

Then, following the algorithm in Figure 4, it is computed the set of disclosable credentials from the disclosure policy and the user's available credentials, and the

minimal set of credentials, out of those, that satisfies the request. The resulting set is $\{\mathsf{credential}(\mathsf{User}: JohnMilburk, \mathsf{Role}: juniorResearcher)\}$. Then the need for this credential is return back to the user.

Example 6. 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.

So, the algorithm updates the user's session profile and the outcome is the need for credential $\mathsf{credential}(\mathsf{User}: JohnMilburk, \mathsf{Role}: seniorResearcher)$.

5 Stateful AC: Missing and Excessing Credentials

What happens if access to services is determined also by the history of past executions? For instance in the example by Atluri and Bertino [8–pag.67] a branch manager of a bank clearing a cheque cannot be the same member of staff who has emitted the cheque. So, if we have no memory of past credentials then it is impossible to enforce any security policy for separation of duties on the application workflow. The problems are the following:

- the request may be inconsistent with some role used by the client in the past;
- the new set of credential may be inconsistent with requirements such as separation of duties;
- in contrast to intra-enterprise workflow systems [8], the partner offering the service has no way to assign to the client the right set of credentials which would be consisted with his future requests (because he cannot assign him future tasks).

So, we must have some roll-back procedure by which, if the user has by chance sent the "wrong" credentials, he can revoke them.

Our interactive access control solution for stateful services and applications is shown in Figure 5.

The logical explanation of the algorithm is the following. Initially when a client requests a specific service the authorization mechanism creates a new session with global variables declined credentials $\mathcal{C}_{\mathcal{N}}$, not revoked credentials $\mathcal{C}_{\mathcal{U}}$, missing credentials $\mathcal{C}_{\mathcal{M}}$ and excessing credentials $\mathcal{C}_{\mathcal{E}}$ set up to empty sets. Then once the session is started, internally, the algorithm loads the policies for access and disclosure control $\mathcal{P}_{\mathcal{A}}$ and $\mathcal{P}_{\mathcal{D}}$ together with the two external sets history of execusion $\mathcal{H}$ and client's active credentials $\mathcal{C}_{\mathcal{P}}$.

Following that, the first step in the algorithm is to get the client's input as sets of currently presented credentials $\mathcal{C}_p$, the revoked ones $\mathcal{C}_r$ and the service request r. Then the set of active credentials $\mathcal{C}_{\mathcal{P}}$ is updated as removing the set $\mathcal{C}_r$ from it and then adding the set of currently presented credentials (rf. step 2). Then in step 3 declined credentials $\mathcal{C}_{\mathcal{N}}$ are updated as credentials the client was asked in the last interactions minus the ones that he has currently presented. Analogously, in step 4, not revoked credentials $\mathcal{C}_{\mathcal{U}}$ are updated as the excessing

Global vars: $\mathcal{C}_\mathcal{N}$, $\mathcal{C}_\mathcal{U}$, $\mathcal{C}_\mathcal{M}$, $\mathcal{C}_\mathcal{E}$; Initially $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{U} = \mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{E} = \emptyset$;
Internal input: $\mathcal{P}_\mathcal{A}$, $\mathcal{P}_\mathcal{D}$, $\mathcal{H}$, $\mathcal{C}_\mathcal{P}$;
Output: grant/deny/$< ask(\mathcal{C}_\mathcal{M}), revoke(\mathcal{C}_\mathcal{E}) >$;

1. **client's input**: $\mathcal{C}_p$, $\mathcal{C}_r$ and r,
2. update $\mathcal{C}_\mathcal{P} = (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_r) \cup \mathcal{C}_p$,
3. update $\mathcal{C}_\mathcal{N} = (\mathcal{C}_\mathcal{N} \cup \mathcal{C}_\mathcal{M}) \setminus \mathcal{C}_p$, where $\mathcal{C}_\mathcal{M}$ is from the last interaction,
4. update $\mathcal{C}_\mathcal{U} = (\mathcal{C}_\mathcal{U} \cup \mathcal{C}_\mathcal{E}) \setminus \mathcal{C}_r$, where $\mathcal{C}_\mathcal{E}$ is from the last interaction,
5. Set up $\mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{E} = \emptyset$,
6. 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 $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \not\models \bot$,
7. if the check succeeds then return ***grant*** else
   (a) compute the set of *disclosable credentials* $\mathcal{C}_\mathcal{D} = \{c \mid \mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{P} \models c\} \setminus (\mathcal{C}_\mathcal{N} \cup \mathcal{C}_\mathcal{P})$,
   (b) use abduction to find a minimal set of *missing credentials* $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ such that both $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \not\models \bot$,
   (c) if a set $\mathcal{C}_\mathcal{M}$ exists then return $<$ ***ask***$(\mathcal{C}_\mathcal{M})$, ***revoke***$(\mathcal{C}_\mathcal{E}) >$ else
      i. use abduction to find a minimal set of missing credentials $\mathcal{C}_\mathcal{M} \subseteq (\mathcal{C}_\mathcal{D} \cup \mathcal{C}_\mathcal{P})$ such that $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{M} \models r$, $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{M} \not\models \bot$ and $\mathcal{C}_\mathcal{U} \cap (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{M}) = \emptyset$,
      ii. if no set $\mathcal{C}_\mathcal{M}$ exists then return ***deny*** else
      iii. compute $\mathcal{C}_\mathcal{E} = \mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{M}$ and $\mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{M} \setminus \mathcal{C}_\mathcal{P}$,
      iv. return $<$ ***ask***$(\mathcal{C}_\mathcal{M})$, ***revoke***$(\mathcal{C}_\mathcal{E}) >$ and iterate.

Fig. 5. Interactive Access Control Algorithm for Stateful Autonomic Services

credentials asked in the last interaction minus the ones currently revoked. Step 5 prepares the two sets $\mathcal{C}_\mathcal{M}$ and $\mathcal{C}_\mathcal{E}$ for the interaction output.

Steps 6, 7, 7a, 7b and 7c have the same explanation as the respective ones in Figure 4. If a set of missing credentials was not found in step 7b then we run the abduction process again (step 7(c)i) but over the extended set of disclosable credentials and active credentials $\mathcal{C}_\mathcal{D} \cup \mathcal{C}_\mathcal{P}$ searching for a solution for r that preserves consistency in $\mathcal{P}_\mathcal{A}$ and unlocks r. The last requirement in the step is used to filter out those solutions that have been partially refused to be revoked.

Step 7(c)i indicates that if a set $\mathcal{C}_\mathcal{M}$ exists then definitely there are "wrong" credentials among those in $\mathcal{C}_\mathcal{P}$ that ban the client to get a solution for r (in step 7b). If no such set then the client is denied because he does not have enough privileges to disclose more credentials to obtain the service r (step 7(c)ii).

Step 7(c)iii computes the sets of excessing and missing credentials $\mathcal{C}_\mathcal{E}$ and $\mathcal{C}_\mathcal{M}$. The motivation behind $\mathcal{C}_\mathcal{E}$ is that the set difference of active credentials minus just computed $\mathcal{C}_\mathcal{M}$ certainly contains the credentials that ban the client to get a solution for r.

At this point there two main issues concerning the set $\mathcal{C}_\mathcal{E}$: (i) the system may restart from scratch asking the client to revoke all his active credentials, i.e. $\mathcal{C}_\mathcal{E} = \mathcal{C}_\mathcal{P}$, (ii) the system may ask the client to present credentials that have been already asked for revokation in past interactions.

Remark 2. Step 7(c)iii looks the opposite of abduction: rather than adding new information to derive more things (the request), we drop information to

derive less things (the inconsistency). One can show that the two tasks are equivalent.

6 Life Is Complicated: Two-Party Negotiation

So far deduction helps us to infer whether a service request is granted by the partner's access policy and the client's set of credentials. In the case of failure, abduction infers what is missing so that the client can still get the desired service.

Example 7. When the senior researcher received the counter request to present his *seniorResearcher* certificate in order to get access he may not want to reveal his role if he is not sure that he talks with a University of Trento's server.

We should allow him to request the system to show a certificate. The system, in its turn, may have policy saying that such certificates are disclosed only to entities coming from an authorized network, e.g., $\mathsf{authNetwork}\,(*, *.fraunhofer.de)$.

The next step is how to establish and automate a two-party negotiation process using the inference capabilities on both sides. For that purpose we need to extend each of the party's policies:

- a policy for access to *own* resources $\mathcal{P}_{\mathcal{AR}}$ on the basis of *foreign* credentials,
- a policy for access to *own* credentials $\mathcal{P}_{\mathcal{AC}}$ on the basis of *foreign* credentials,
- a policy for the disclosure of the need of missing *foreign credentials* $\mathcal{P}_{\mathcal{D}}$.

Client and the server just have to run the same negotiation protocol:

1. The client, Alice, sends a service request r and (optionally) a set of credentials $\mathcal{C}_p$ to the server, Bob.
2. Then Bob looks at r and if it is a request for a service he calls the interactive access control algorithm in Figure 4 with his policies for access and disclosure of resources $< \mathcal{P}_{\mathcal{AR}}, \mathcal{P}_{\mathcal{D}} >$.
3. If r is a request for a credential then he calls the same algorithm with his respective policies for access and disclosure of credentials $< \mathcal{P}_{\mathcal{AC}}, \mathcal{P}_{\mathcal{D}} >$.
4. In the case of computed missing credentials $\mathcal{C}_{\mathcal{M}}$, he transforms that into counter-requests for credentials and waits until receives all responses. At this point Bob acts as a client, requesting Alice the set of credentials $\mathcal{C}_{\mathcal{M}}$. Alice will run the same protocol swapping roles.
5. When Bob's main process receives all responses it checks whether the missing credentials have been supplied by Alice.
6. If $\mathcal{C}_{\mathcal{M}}$ was not reached, Bob restarts the loop and consults the interactive access control algorithm for a new decision.
7. When a final decision is taken, the response (grant/deny) is sent to Alice.

The protocol can be run on both sides so that they can communicate and negotiate the missing credentials until enough trust is established and the service is granted or the negotiation failed and the process is terminated.

7 Induction: Finding the Rules

The work for inductive logic programming [14] has been most evolved in the field of machine learning. Inductive logic programming systems (ILP) construct concept definitions from examples and a logical domain theory.

Induction may be an extremnely valuable tool for autonomic nodes, because complete and consistent access policies may be difficult to write. So it might be weel the case that a node has only a partial policy, and some additional set of examples of access that one desired to permit or forbid. Then the node should be able, by generalizing from the examples to derive a policy that matched the given examples and is also asble to answer other similar queries.

So an autonomic node could be provided with background police P_B, some sample granted requests for services R^+ and denied requests R^- and as a result it should be able to constructs a tentative access policy P_{HA}. Here R^+, R^- are sets of ground facts and P_B and P_{HA} are logic programs. The conditions for construction of P_{HA} are:

Necessity: $P_B \not\models R^+$,
Sufficiency: $P_B \wedge P_{HA} \models E^+$,
Weak consistency: $P_B \wedge P_{HA} \not\models \bot$,
Strong consistency: $P_B \wedge P_{HA} \wedge E^- \not\models \bot$.

A number of algorithms can be used for determining the construction of P_{HA} based on ILP (see e.g. [14]) and the identification of the most appropriate for autonomic communication policies is the subject of future work.

8 Implementation

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

For our implementation, Collaxa[1] is used as a main manager of Web Services Business Processes (on the AuthorizationServer side).

PolicyEvaluator is a Java module that acts as a wrapper for the DLV system[2] (a disjunctive datalog system with negations and constraints) and implements our interactive algorithm for stateless autonomic nodes (Fig. 4). For deductive computations we use the disjunctive datalog front-end (the default one) while for abductive computations, the diagnosis front-end.

The current system processes credentials at an high level: defines what can be inferred and what is missing from a partner's access policy and a user's set of credentials. For the actual distributed management of credentials at lower levels (namely actual cryptographic verification of credentials) we decided to use

[1] Collaxa BPEL Server (v2.0 rc3) – www.collaxa.com
[2] DLV System (r. 2003-05-16) – www.dlvsystem.com

PERMIS infrastructure [18] because it incorporates and deals entirely with X.509 Identity and Attribute Certificates. It allows for creating, allocating, storing and validating such certificates. Since PERMIS conforms to well-defined standards we can easily interoperate with the other entities (partners) in the network.

9 The Challenges Ahead

So far we have presented a logical framework and a proof-of-concept implementation for reasoning about access control for autonomic communication based on interaction for supplying missing credentials or for revoking "wrong" credentials. We have discussed the different formal reasoning services – deduction, abduction, and induction, with a special emphasis of the first two. We have also show how the model can deal with stateful access to services and two party negotiation.

Yet, a number of major challenges remains:

Complexity Characterization: abduction engines such as DLV are rather effective but unfortunately general algorithms for abduction are inefficient[3]. 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). So capturing the exact computational complexity of the problem may be far from trivial.

Approximation vs Language Restriction: even if the problem is hard in the general case, we might have suitable syntactic restrictions that allow for a polynomial evaluation. In other cases, we may be able to find out anytime algorithms that gives an approximate answer (not really the minimal one but close to it).

Reputation Management: so far we have assumed that declining or presenting a credential has no impact on the reputation of nodes. Research on algorithms and logics for secure reputation is still in the early stage but its integration with interactive access control might have a significant impact.

Negotiatation Strategy Analysis: which is the impact of the negotiation strategy on the effectiveness, completeness, privacy protection, immunity from DoS attacks of interactive access control? So far only the completeness of the procedure is settled and more sophisticated strategies, taking into account the value of credentials that are disclosed could lead to many interesting results relevant for the practical deployment of the framework.

Policy Compilation: this is likely the topic with major impact on industry. All policies (either in networking or security) are either interpreted or hardwired in the application. In contrast, we would need a way to "compile" the policy and the policy enforcement engine into machine languages so that autonomic nodes can quickly react to the requests and yet gives us the flexibility of policies: update a policy simply means recompiling and redeploying.

[3] They lay at the secon level of the polynomial hierarchy, i.e. harder than NP.

References

1. Sloman, M., Lupu, E.: Policy specification for programmable networks. In: 1st Inter. Working Conference on Active Networks, Springer-Verlag (1999) 73–84
2. 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), Springer (2003) 135–146
3. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder policy specification language. In: Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY), Springer-Verlag (2001) 18–38
4. Weeks, S.: Understanding trust management systems. In: IEEE Symposium on Security and Privacy (SS&P), IEEE Press (2001)
5. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B.M., Ylonen, T.: SPKI Certificate Theory. (1999) IETF RFC 2693.
6. Li, N., Grosof, B.N., Feigenbaum, J.: Delegation logic: A logic-based approach to distributed authorization. ACM TISSEC **6** (2003) 128–171
7. Atluri, V., Chun, S.A., Mazzoleni, P.: A Chinese wall security model for decentralized workflow systems. In: Proceedings of the 8th ACM CCS. (2001) 48–57
8. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. ACM TISSEC **2** (1999) 65–104
9. Georgakopoulos, D., Hornick, M.F., Sheth, A.P.: An overview of workflow management: From process modeling to workflow automation infrastructure. Distributed and Parallel Databases **3** (1995) 119–153
10. Kang, M.H., Park, J.S., Froscher, J.N.: Access control mechanisms for inter-organizational workflow. In: 6th ACM SACMAT. (2001) 66–74
11. Bertino, E., Catania, B., Ferrari, E., Perlasca, P.: A logical framework for reasoning about access control models. In: 6th ACM SACMAT. (2001) 41–52
12. Bonatti, P., Samarati, P.: A unified framework for regulating access and information release on the web. Journal of Computer Security **10** (2002) 241–272
13. Shanahan, M.: Prediction is deduction but explanation is abduction. In: Proceedings of IJCAI '89, Morgan Kaufmann (1989) 1055–1060
14. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. JLP **19/20** (1994) 629–679
15. Apt, K.: Logic programming. In van Leeuwen, J., ed.: Handbook of Theoretical Computer Science. Elsevier (1990)
16. De Capitani di Vimercati, S., Samarati, P.: Access control: Policies, models, and mechanism. In Focardi, R., Gorrieri, F., eds.: Foundations of Security Analysis and Design - Tutorial Lectures. Volume 2171 of LNCS. Springer Verlag Press (2001)
17. Koshutanski, H., Massacci, F.: Interactive access control for Web Services. In: 19th IFIP Information Security Conference (SEC), Kluwer Press (2004) 151–166
18. Chadwick, D.W., Otenko, A.: The PERMIS X.509 role-based privilege management infrastructure. In: Seventh ACM SACMAT. (2002) 135–140