Journal of Network and Systems Management, Vol. 15, No. 1, March 2007 (© 2007)
DOI: 10.1007/s10922-006-9057-2

A Negotiation Scheme for Access Rights Establishment in Autonomic Communication

Hristo Koshutanski[1] and Fabio Massacci[2]

Published online: 2 March 2007

Autonomic computing and communication has become a new paradigm for dynamic service integration and resource sharing in today's ambient networks. Devices and systems need to dynamically collaborate and federate with little known or even unknown parties in order to perform everyday tasks. Those devices and systems act as independent nodes that autonomously manage and enforce their own security policies.

Thus in autonomic pervasive communications clients may not know a priori what access rights they need in order to execute a service nor service providers know a priori what credentials and privacy requirements clients have so that they can take appropriate access decisions.

To solve this problem we propose a negotiation scheme that protects security and privacy interests with respect to information disclosure while still providing effective access control to services. The scheme proposes a negotiation protocol that allows entities in a network to mutually establish sufficient access rights needed to grant a service.

KEY WORDS: access control; policy based access management; trust negotiation; logic reasoning; algorithms; protocols.

1. INTRODUCTION

The last decade has seen an exponentially growth of mobile devices that communicate with a variety of sources and infrastructures in order to form coherent communication networks suitable for their needs and tasks.

A major issue in this scenario is that communications occur between autonomic nodes that have independent management and enforcement of their own

[1]CREATE-NET, Via Solteri 38, University of Malaga, Trento 38100, Italy. E-mail: hristo@lcc.uma.es.
[2]Dip. di Informatica e Telecomunicazioni, Univ. di Trento, Via Sommarive 14, 38050 Povo di Trento, Italy. E-mail: fabio.massacci@unitn.it.

security policies. Access requests and access decisions must be taken autonomously with incomplete information about the partners.

In an autonomic communication scenario a client may have all the necessary credentials to access a service but may simply not know it. Equally, it is unrealistic to assume that servers will make public available their security policies so that clients can do a policy evaluation themselves and just come out with just the right credential for access.

Therefore, autonomic servers should be able to ask clients on-the-fly for additional credentials that are sufficient to grant access. On the other side, autonomic clients, once asked for additional credentials, should be able to evaluate their own polices and counter-request servers for some evidence in order to establish enough confidence to disclose the missing credentials.

The usage of security policies has been already a major paradigm shift in access control to communication services over the last few years. Indeed, one may speak of *policy-based self-management* of services (e.g., [1, 2] or the IEEE Policy Workshop series). The intuition is that access to services and resources is automatically derived from policies. Devices look at the requested action and credentials presented to them, evaluate the policy rules according to the new facts and derive the allowed actions [1, 3]. Autonomic communication brings us beyond policy based access control. How can partners negotiate access without knowing each other's security policies?

1.1 The Contribution of this Paper

This paper shows how one can bootstrap from a simple rule-based policy framework and using some advanced reasoning services a comprehensive access negotiation scheme for autonomic communication. The negotiation scheme protects privacy and security interests with respect to information disclosure and access control on client and server sides. It allows two entities in a network to mutually establish access rights agreement needed to proceed with the service request. It supports hierarchical policies (indeed arbitrary policies for which reasoning services exist), non-monotone reasoning, and fine granularity control on the disclosure of information.

The rest of paper is organized as follows. Section 2 introduces the intuitive description of interactive access control model and the advanced reasoning services. Section 3 presents the underlying policy model and formally defines the advanced reasoning services. The interactive access control protocol is shown in Section 4. Next, Section 5 outlines the negotiation policy framework. The negotiation protocol is presented in Section 6 followed by the stepwise disclosure algorithm described in Section 7. Finally, Section 8 concludes the paper and outlines future work.

1. check whether $\mathcal{P}_\mathcal{A}$ and $\mathcal{C}_p$ entail r,
2. if the check succeeds then `grant` access
3. else `deny` access.

Fig. 1. Traditional access control.

2. INTERACTIVE ACCESS CONTROL

We will introduce step-by-step the concept of interactive access control by "evolving" existing access control frameworks.

Let us start with the traditional access control. A server has a *security policy for access control* $\mathcal{P}_\mathcal{A}$ that is used when taking decisions about the usage of services offered by a service provider. A user submits a set of credentials $\mathcal{C}_\mathcal{P}$ and a service request r in order to execute a service. We say that policy $\mathcal{P}_\mathcal{A}$ and credentials $\mathcal{C}_\mathcal{P}$ entail r meaning that request r should be granted by the policy $\mathcal{P}_\mathcal{A}$ and the presented credentials $\mathcal{C}_\mathcal{P}$.

Figure 1 shows the "traditional" access control decision process [4]. Whether the decision process uses Role-Based Access Controls (RBAC) [5], Simple Public Key Infrastructure (SPKI) [6], RT role-based trust-management framework [7] or any other trust management framework it is immaterial at this stage: they can be captured by suitably defining $\mathcal{P}_\mathcal{A}$, $\mathcal{C}_\mathcal{P}$ and the entailment operator.

A number of works has deemed such blunt denials unsatisfactory. Bonatti and Samarati [8] and Yu *et al*. [9] proposed to send back to clients some of the rules that are necessary to gain additional access. Figure 2 shows the essence of the approaches.

Both works impose several syntactical restrictions on the format of the policy and essentially merge two different security issues: the policy for governing access to server's *own resources* and the policy for governing disclosure of the need, by the server, of *foreign credentials* (from the client).

Still, the foremost limitation is that both approaches require policies to be flat: a policy protecting a resource must contain all credentials needed to allow

1. check whether $\mathcal{P}_\mathcal{A}$ and $\mathcal{C}_p$ entail r,
2. if the check succeeds then `grant` access else
   (a) find a rule $r \leftarrow p \in \mathit{PartialEvaluation}(\mathcal{P}_\mathcal{A} \cup \mathcal{C}_p)$, where p is a partial policy protecting r,
   (b) if such a rule exists then send it back to the client else deny access.

Fig. 2. Disclosable access control.

access to that resource. As a result, it calls for structuring of policy rules that is counter-intuitive from the access control point of view. For instance, a policy rule may say that for access to the full text of an on-line journal article a requester must satisfy the requirements for browsing the journal's table of contents plus some additional credentials. A rule detailing access to the table of contents could then specify another set of credentials. Even this simple scenario is not allowed in either formalisms.

Yet, constraints that would make policy reasoning non-monotone (such as separation of duties) require to look at more than one rule at a time. So, if the policy is not flat, it has constraints on the credentials that can be presented at the same time (e.g., separation of duties) or a more complex role hierarchy is used, these systems would not be complete.

Bonatti and Samarati's approach has further limitations on the granularity level of disclosure of information. In their work, governing access to a service is composed in two parts: a prerequisite rule and a requisite rule. Prerequisite rules specify the requirements that a client should satisfy before being considered for the requirements stated by the requisite rules, which in turn grant access to services. Thus, prerequisite rules play the role of controlling the disclosure of the service requisite rules. In this way their approach does not decouple policy disclosure from policy satisfaction, as already noted by Yu and Winslett [10].

The work by Yu and Winslett [10] overcomes this latter limitation and proposes to treat policies as first class resources, i.e., each policy protecting a resource is considered as a sensitive resource itself whose disclosure is recursively protected by another policy. Still they have the same flatness, unicity and monotonicity limitations. This happens because the only reasoning service used so far for access control is *deduction* – check whether the request follows from the policy and the presented credentials.

We claim that we need another, less-known, reasoning service *abduction* – check what missing credentials are necessary so that the request can follow from the policy and the presented credentials. Thereupon, we present the basic idea of *interactive access control* shown in Fig. 3.

1. check whether $\mathcal{P}_{\mathcal{A}}$ and C_p entail r,
2. if the check succeeds then `grant` access else

(a) compute a set $C_{\mathcal{M}}$ such that:
- $\mathcal{P}_{\mathcal{A}}$ together with C_p and $C_{\mathcal{M}}$ entail r, and
- $\mathcal{P}_{\mathcal{A}}$ together with C_p and $C_{\mathcal{M}}$ preserve consistency.

(b) if $C_{\mathcal{M}}$ exists then ask the client for $C_{\mathcal{M}}$ and iterate

(c) else `deny` access.

Fig. 3. Basic idea of interactive access control.

The "compute a set $\mathcal{C}_{\mathcal{M}}$ such that..." (step 2a) is exactly the operation of abduction. This solution raises a new challenge: how do we decide the potential set of missing credentials? It is clearly undesirable to disclose all credentials occurring in $\mathcal{P}_{\mathcal{A}}$ and, therefore, we need a way to define how to control the disclosure of such a set.

In [10] Yu and Winslett addressed partly this issue by protecting policies within the access policy itself. They argue that policies for protecting resources should be themselves treated as first class sensitive resources. The authors distinguish between policy disclosure and policy satisfaction which allows them to have control on when a policy can be disclosed from when a policy is satisfied. However, this is not really satisfactory as it does not decouple the decision about access from the decision about disclosure.

So, we need *two* policies: one for granting access to one's own resources and one for disclosing the need of foreign (someone else's) credentials. Therefore, we introduce a *security policy for disclosure control* $\mathcal{P}_{\mathcal{D}}$. The policy for disclosure control identifies the 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 occurring in $\mathcal{P}_{\mathcal{A}}$ are disclosable to (i.e. can be asked to) the requestor.

Yu and Winslett policies determine whether a client is authorized to be informed of the need to satisfy a given policy. While, in our case, having a separate disclosure policy $\mathcal{P}_{\mathcal{D}}$ allows us to have a finer-grained disclosure control over the information flow back to a client. Instead of controlling the disclosure of (entire) policies as a finest-grained unit we are able to control the disclosure of single credentials, composing those policies, separately and independently from the disclosure of the policies themselves.

The refined algorithm for *interactive access control with controlled disclosure* is in Fig. 4.

1. check whether $\mathcal{P}_{\mathcal{A}}$ and C_p entail r,
2. if the check succeeds then `grant` access else
   (a) compute the set of disclosable credentials $C_{\mathcal{D}}$ entailed by $\mathcal{P}_{\mathcal{D}}$ and C_p,
   (b) compute a set $C_{\mathcal{M}}$ out of the disclosable ones ($C_{\mathcal{M}} \subseteq C_{\mathcal{D}}$) such that
   – $\mathcal{P}_{\mathcal{A}}$ together with C_p and $C_{\mathcal{M}}$ entail r, and
   – $\mathcal{P}_{\mathcal{A}}$ together with C_p and $C_{\mathcal{M}}$ preserve consistency.
   (c) if $C_{\mathcal{M}}$ exists then ask the client for $C_{\mathcal{M}}$ and iterate
   (d) else `deny` access.

Fig. 4. Interactive access control with controlled disclosure.

Example 1. Example [10] formalized as two logic programs:

$$\mathcal{P}_{\mathcal{D}}: \left| \begin{array}{l} C_{AliceID} \\ C_{CSWL} \leftarrow C_{McKinleyEmployee} \\ C_{RoI} \leftarrow C_{McKinleyEmployee} \end{array} \right. \qquad \mathcal{P}_{\mathcal{A}}: \left| \begin{array}{l} r \leftarrow C_{AliceID} \\ r \leftarrow C_{CSWL}, C_{RoI} \end{array} \right.$$

$\mathcal{P}_{\mathcal{D}}$ states that the disclosure of Alice's ID is not protected and potentially released to anybody. The need for credentials California social worker license C_{CSWL} and release-of-information C_{RoI} is disclosed only to users who have already presented their McKinley employee certificate $C_{McKinleyEmployee}$.

$\mathcal{P}_{\mathcal{A}}$ states that access to r is granted either to Alice or to California social workers that have a release-of-information credential issued by Alice.

There is still a tricky question to be answered: How do we know that the algorithm terminates? In other words, how do we know that a client can actually arrive to a grant? For example, can we assure that the server will not keep asking Alice for a full professor credential which she does not have, while never asking for a senior researcher credential, which she has?

We refer the reader to [11–13] for details on the interactive access control framework and its properties of completeness and correctness.

3. THE UNDERLYING LOGICAL MODEL

Access and disclosure policies are written as normal logic programs [14]. A normal logic program is a set of *rules* of the form:

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

where A, B_i and C_i are (possibly ground) predicates. 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 B_i and *not* C_j is called the *body* of the rule. If the body (resp. head) is empty the rule is a *fact* (resp. *constraint*).

The semantics used in the model is the stable model semantics [15] (see also [14] 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.

Now we can formally define the reasoning services.

Definition 1. Let $\mathcal{P}$ be a policy and L be a ground literal. L is a *consequence* of $\mathcal{P}$, $\mathcal{P} \models L$, if L is true in every stable model of $\mathcal{P}$. $\mathcal{P}$ is *consistent*, $\mathcal{P} \not\models \bot$, if there is a stable model for $\mathcal{P}$.

Definition 2. A resource r is a *security consequence* of a policy $\mathcal{P}$ if (i) $\mathcal{P}$ is consistent and (ii) r is a consequence of $\mathcal{P}$.

Definition 3. Let $\mathcal{P}$ is a policy and r be a resource. A set of credentials $\mathcal{C}_S$ is a *solution set* for r according to $\mathcal{P}$ if r is a security consequence of $\mathcal{P}$ and $\mathcal{C}_S$, i.e. $\mathcal{P} \cup \mathcal{C}_S \models r$ *and* $\mathcal{P} \cup \mathcal{C}_S \neq \bot$.

Definition 4. A policy $\mathcal{P}$ is *monotonic* if whenever a set of statements $\mathcal{C}_S$ is a solution set for r according to $\mathcal{P}(\mathcal{P} \cup \mathcal{C}_S \models r)$ then any superset $\mathcal{C'}_S \supset \mathcal{C}_S$ is also a solution set for r according to $\mathcal{P}(\mathcal{P} \cup \mathcal{C'}_S \models r)$.

Definition 5. Let $\mathcal{P}$ be a policy, $\mathcal{A}$ be a set of ground atoms (credentials) and L be a positive literal. L is *one-step deducible* from (consequence of) $\mathcal{A}$ according to $\mathcal{P}$, $\mathcal{A} \models_1^{\mathcal{P}} L$, if for some literals $L_1,\ldots, L_n$ holds:

(i) $L \leftarrow L_1, \ldots, L_n$, is in $\texttt{ground}(P \cup \mathcal{A})$[3] and
(ii) for all credential literals $L_{c1}, \ldots, L_{cp}$, $1 \leq ci \leq n$, holds: (a) if L_{ci} is a positive literal then $\mathcal{A} \models L_{ci}$, (b) if L_{ci} is a negative literal then $\mathcal{P} \cup \mathcal{A}| = L_{ci}$.

Definition 6. Let $\mathcal{P}$ be a policy, $\mathcal{H}$ a set of ground atoms, L a ground literal, and $\prec$ a partial order (p.o.) over subsets of $\mathcal{H}$. A solution of the abduction problem $\langle L, \mathcal{H}, \mathcal{P}\rangle$ is a set of ground atoms $\mathcal{E}$ such that: (i) $\mathcal{E} \subseteq \mathcal{H}$, (ii) $\mathcal{P} \cup \mathcal{E} \models L$, (iii) $\mathcal{P} \cup \mathcal{E} \not\models \bot$, (iv) any set $\mathcal{E}' \prec \mathcal{E}$ does not satisfy all conditions above.

Traditional partial orders are subset containment or set cardinality.

4. THE ACCESS CONTROL PROTOCOL

Below we summarize all the information we need to state the protocol.
$\mathcal{P}_{\mathcal{A}}$ – the policy governing access to resources,
$\mathcal{P}_{\mathcal{D}}$ – the policy controlling the disclosure of foreign (missing) credentials,
$\mathcal{C}_p$ – set of credentials presented by a client in a single interaction,
$\mathcal{C}_{\mathcal{P}}$ – set of active credentials that have been presented by a client during an interactive access control process,
$\mathcal{C}_{\mathcal{N}}$ – set of credentials that a client has declined to present during an interactive access control process.

The set of declined credentials $\mathcal{C}_{\mathcal{N}}$ assures termination. Figure 5 shows the interactive access control decision algorithm and protocol.

The intuition behind the algorithm is the following. Once the client has initiated a service request r, possibly with a set of credentials $\mathcal{C}_p$, the interactive

[3]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 [14].

Input: r, C_p
Output: grant/deny/ask($C_{\mathcal{M}}$)
i`AccessControl`(r, C_p)

1: $C_{\mathcal{P}} = C_{\mathcal{P}} \cup C_p$;
2: $C_{\mathcal{N}} = C_{\mathcal{N}} \cup (C_{\mathcal{M}} \setminus C_p)$, where $C_{\mathcal{M}}$ is from the last interaction;
3: *result* = *i*`AccessDecision`(r, $\mathcal{P}_{\mathcal{A}}$, $\mathcal{P}_{\mathcal{D}}$, $C_{\mathcal{P}}$, $C_{\mathcal{N}}$);
4: return *result*;

i`AccessDecision`(r,$\mathcal{P}_{\mathcal{A}}$,$\mathcal{P}_{\mathcal{D}}$,$C_{\mathcal{P}}$,$C_{\mathcal{N}}$)

1. check whether r is a security consequence of $\mathcal{P}_{\mathcal{A}}$ and $C_{\mathcal{P}}$, namely
   – $\mathcal{P}_{\mathcal{A}} \cup C_{\mathcal{P}} \models r$, and
   – $\mathcal{P}_{\mathcal{A}} \cup C_{\mathcal{P}} \not\models \bot$.
2. if the check succeeds then return ***grant*** else
   (a) compute the set of disclosable credentials $C_{\mathcal{D}}$ as
   $C_{\mathcal{D}} = \{ c \mid c$ credential that $\mathcal{P}_{\mathcal{D}} \cup C_{\mathcal{P}} \models c \} \setminus (C_{\mathcal{N}} \cup C_{\mathcal{P}})$,
   (b) use abduction to find a set of missing credentials $C_{\mathcal{M}} (\subseteq C_{\mathcal{D}})$ such that:
   – $\mathcal{P}_{\mathcal{A}} \cup C_{\mathcal{P}} \cup C_{\mathcal{M}} \models r$, and
   – $\mathcal{P}_{\mathcal{A}} \cup C_{\mathcal{P}} \cup C_{\mathcal{M}} \not\models \bot$.
   (c) if no such set exists then return ***deny*** else
   (d) return ***ask***($C_{\mathcal{M}}$).

Fig. 5. Interactive access control algorithm and protocol.

algorithm updates the client's profile of $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{N}}$ (lines 1: and 2:). $\mathcal{C}_{\mathcal{P}}$ is updated with the newly presented credentials $\mathcal{C}_p$. $\mathcal{C}_{\mathcal{N}}$ is updated with the set difference of what the client was asked in the last interaction ($\mathcal{C}_{\mathcal{M}}$) minus what he presents in the current one $\mathcal{C}_p$.

Next, the algorithm takes an access decision (line 3:). The first step of the access decision function is to check whether the request r is granted by $\mathcal{P}_{\mathcal{A}}$ according to the client's set $\mathcal{C}_{\mathcal{P}}$ (step 1). If the check fails, the starting point of the interactive framework, then in step 2a, 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. The latter is used to avoid repeatedly asking something already declined or presented. Then, the algorithm computes (using abduction) all subsets of $\mathcal{C}_{\mathcal{D}}$ that are consistent with the access policy $\mathcal{P}_{\mathcal{A}}$ and grant r. Out of those sets (if any) the algorithm selects the minimal one.

Example 2. (rf. [13] for details) A senior researcher at Fraunhofer institute FOKUS wants to reconfigure an online service for paper submissions for a workshop. The service is part of a big management system of the Planet-Lab network.

So, for doing that, at the time of access, she presents her employee certificate, issued by a Fraunhofer certificate authority, presuming that it is enough as a potential customer, formally speaking `credential(aliceMilburk, employee).`

According to the access policy the credentials are not enough to get configure access. Then, following the algorithm, abduction reasoning computes the following sets of credentials that satisfy the request:

```
{credential(aliceMilburk, juniorResearcher)},

{credential(aliceMilburk, seniorResearcher)},

{credential(aliceMilburk, boardOfDirectors)}.
```

Then, using role minimality criterion, the algorithm returns back the need for

```
credential(aliceMilburk, juniorResearcher).
```

Since Alice is a senior researcher, she declines to present the requested credential by returning an empty set of presented credentials. The algorithm updates the client's profile by marking the requested credential as declined. Next, the algorithm recomputes the missing sets of credentials:

```
{credential(aliceMilburk, seniorResearcher)},

{credential(aliceMilburk, boardOfDirectors)}.
```

and returns the need for `credential(aliceMilburk, seniorResearcher)` back to the client. Then on the next interaction Alice presents her certificate for a senior researcher and the algorithm grants the service request.

5. THE NEGOTIATION SCHEME

Let us consider the following autonomic communication scenario:

1. Alice wants to access some service of Bob.
2. Alice does not know exactly what credentials Bob needs, so (a) Bob must compute what is missing and ask Alice, (b) Alice must send to Bob all credentials he requested.
3. In response to (2b), Alice may want to have some credentials from Bob before sending hers, so (a) she must tell Bob what he needs to provide, (b) Bob must have a policy to decide how access to his credentials is granted.
4. In response to (2a), Bob may not want to disclose all that is missing at once but may want to ask Alice first some of the less sensitive credentials, so Bob must have a way to request in a stepwise fashion the missing credentials.

To combine automated trust negotiation and interactive access control we assume that clients and servers have the three security policies:

1. $\mathcal{P}_{\mathcal{AR}}$: policy for protecting *own* resources based on *foreign* credentials
2. $\mathcal{P}_{\mathcal{AC}}$: policy for protecting *own* credentials based on *foreign* credentials
3. $\mathcal{P}_{\mathcal{D}}$: policy for disclosure the need of (missing) *foreign* credentials

Technically speaking we could merge policies $\mathcal{P}_{\mathcal{AR}}$ and $\mathcal{P}_{\mathcal{AC}}$ into a flat policy for protecting sensitive resources as in [9, 10]. However, the structured approach is better because the criteria behind and likely the administrator of each policy are different. Resource access is decided by the business logic whereas credential access is due to security and privacy considerations.

For example the negotiation of a sensitive credential may require activation of credentials that are not considered from the business logic for the actual access control process and even they may be inconsistent with the business logic rules. Thus, forcing separation between policies $\mathcal{P}_{\mathcal{AR}}$ and $\mathcal{P}_{\mathcal{AC}}$ we free the access policy $\mathcal{P}_{\mathcal{AR}}$ to be arbitrarily complex with almost everything that is on the (Datalog) access control market (say with negation as failure, constraints on separation of duties, or other credentials such as those by Li and Mitchell [7]).

Rather, the policy for access to own credentials we restrict to be monotonic because of its particular nature: once access to a credential is agreed (granted) it is agreed! In contrast, a credential for access to resources may come and go due to separation of duty or other constraints.

6. THE NEGOTIATION PROTOCOL

First, we introduce the notation O denoting a set of own credentials with respect to a negotiation opponent. Now, let us recall the `iAccessControlprotocol`, presented in Section 4, adapted to the new convention and with the following modification. Instead of returning the need of missing credentials we transform $\mathcal{C}_{\mathcal{M}}$ into a sequence of single requests each asking for a foreign credential from the missing set. Figure 6 shows the core protocol.

We extend the protocol to work on client and server sides so that they automatically request each other for missing foreign credentials. Step 1 of the protocol updates the set of foreign presented credentials with those presented at the time of request. These presented credentials are typically pushed by the opponent when initiate a service request. After the initial update we go in a loop where `iAccessDecision` algorithm is run for an access decision. The purpose of the loop is to keep asking the opponent new solutions (missing credentials) until a decision of grant or deny is taken.

The technicality of the protocol is in step 6 where we represent the request for a missing credential as a remote invocation of the `iAccessNegotiation` protocol on the opponent side. In this way, the new protocol has the same functionality as

```
Session vars: C_P and C_N. Initially C_P = C_N = ∅.
iAccessNegotiation(r, C_p)
 1: C_P = C_P ∪ C_p;
 2: repeat
 3:    result = iAccessDecision(r, P_A, P_D, C_P, C_N);
 4:    if result == ask(C_M) then
 5:       for each c ∈ C_M do
 6:          response = invoke iAccessNegotiation(c, ∅)@Opponent;
 7:          if response == grant then
 8:             C_P = C_P ∪ { c };
 9:          else
10:             C_N = C_N ∪ { c };
11:       done
12: until result == grant or result == deny.
13: return result;
```

Fig. 6. The core of the negotiation protocol.

the `iAccessControl` protocol if the client just replies whether he has a credential or not.

Step 6 invokes `iAccessNegotiation` protocol with an empty set of presented own credentials. One can modify the protocol by introducing a function `PushedCredentials(`c`)` that decides what own credentials (O_{push}) an opponent has to push when requesting for a foreign credential c. This is an issue for practical implementations and we omit it for the sake of clarity. Details can be found in [11].

For bilateral negotiation we must consider the following issues:

- each request for a credential spurs a new negotiation thread that negotiates access to this credential.
- during a negotiation process parties may start to request credentials from each other that are already in a negotiation. So, the notion of suspended credential requests must be taken into account.

Figure 7 shows the updated version of the `iAccessNegotiation` protocol.

With its new version, whenever a request arrives it is run in a new thread that shares the same session variables $\mathcal{C}_{\mathcal{P}}$, $\mathcal{C}_{\mathcal{N}}$ and O_{neg} with other threads running under the same negotiation process. The set O_{neg} keeps track of the opponent's own credentials that have been requested and which are still in a negotiation.

Now, if a request for a credential, which is already in a negotiation, is received the protocol suspends the new thread until the respective negotiation thread finishes

```
Session vars: C_P, C_N and O_neg. Initially C_P = C_N = O_neg = ∅.
iAccessNegotiation(r, C_p) - runs in a new thread
 1:  C_P = C_P ∪ C_p;
 2:  if r ∈ O_neg then
 3:     suspend and await for the result on r's negotiation;
 4:     return result when resumed;
 5:  else
 6:     O_neg = O_neg ∪ { r };
 7:     repeat
 8:        result = iAccessDecision(r, P_A, P_D, C_P, C_N);
 9:        if result == ask(C_M) then
10:           for each c ∈ C_M do
11:              response = invoke iAccessNegotiation(c, ∅)@Opponent;
12:              if response == grant then
13:                 C_P = C_P ∪ { c };
14:              else
15:                 C_N = C_N ∪ { c };
16:           done
17:        fi
18:     until result == grant or result == deny.
19:     O_neg = O_neg \ { r };
20:     resume all processes awaiting on r with the result of the negotiation;
21:     return result;
22:  endelse
```

Fig. 7. The negotiation protocol with suspended credentials.

(step 3). Then, when the original thread returns an access decision the protocol resumes all threads awaiting on the requested credential and informs them for the final decision (step 20).

Figure 8 shows the complete negotiation protocol. Whenever a service request is received the `iAccessDispatcher` module runs `iAccessNegotiation` in a new session process and initializes $\mathcal{C}_{\mathcal{P}}$, $\mathcal{C}_{\mathcal{N}}$ and O_{neg} to an empty set (step 2). Then each counter-request for a credential is run in a new thread under the same negotiation process (step 4).

On the other hand, whenever an entity requests a service r at the opponent side, presenting initially some own credentials O_p, the `iAccessDispatcher` server invokes `iAccessNegotiation@Opponent` in a new session process so that any counter-request from the opponent is run in a new thread under the same negotiation process.

Session vars: C_P, C_N and O_{neg}. Initially $C_P = C_N = O_{neg} = \varnothing$.

*i*AccessDispatcher

OnReceiveRequest: *i*AccessNegotiation(r, C_p)

1: if isService(r) then
2: reply *i*AccessNegotiation(r, C_p); in a new session process.
3: else
4: reply *i*AccessNegotiation(r, C_p); in a new thread under the original session.

OnSendRequest: < r, O_p >

1: if isService(r) then
2: invoke *i*AccessNegotiation(r, O_p)@Opponent in a new session process.

*i*AccessNegotiation(r, C_p)

1: $C_P = C_P \cup C_p$;
2: if $r \in O_{neg}$ then
3: suspend and await for the *result* on r's negotiation;
4: return *result* when resumed;
5: else
6: $O_{neg} = O_{neg} \cup \{ r \}$;
7: repeat
8: if isService(r) then
9: *result* = *i*AccessDecision(r, $\mathcal{P}_{AR}$, $\mathcal{P}_D$, C_P, C_N);
10: else
11: *result* = *i*AccessDecision(r, $\mathcal{P}_{AC}$, $\mathcal{P}_D$, C_P, C_N);
12: if *result* == ask(C_M) then
13: AskCredentials(C_M);
14: until *result* == grant or *result* == deny;
15: $O_{neg} = O_{neg} \setminus \{ r \}$;
16: resume all processes awaiting on r with the *result* of the negotiation;
17: return *result*;
18: endelse

AskCredentials(C_M)

1: parfor each $c \in C_M$ do
2: *response* = invoke *i*AccessNegotiation(c, $\varnothing$)@Opponent;
3: if *response* == grant then $C_P = C_P \cup \{ c \}$ else $C_N = C_N \cup \{ c \}$;
4: done
5: while $C_M \not\subseteq (C_P \cup C_N)$ do wait(timeout);

Fig. 8. The negotiation protocol.

The intuition behind the negotiation protocol is the following:

1. A client, Alice, sends a service request r and (optionally) a set of (new own) credentials O_p to a server, Bob.
2. Bob's `iAccessDispatcher` receives the requests and runs `iAccessNegotiation`($r, \mathcal{C}_p$) in a new process (with $\mathcal{C}_p = O_p$).
3. Once the protocol is initiated, it updates the over all set of presented foreign credentials with the newly presented ones and checks whether the request should be suspended or not (steps 1 and 2).
4. If not suspended, then Bob looks at r and if it is a request for a service he calls `iAccessDecision` with his *policy for access to resources* $\mathcal{P}_{\mathcal{AR}}$, his disclosure policy $\mathcal{P}_{\mathcal{D}}$, the set of foreign presented credentials $\mathcal{C}_{\mathcal{P}}$ and the set of foreign declined credentials $\mathcal{C}_{\mathcal{N}}$ (step 9).
5. If r is a request for a credential then he calls `iAccessDecision` with his *policy for access to own credentials* $\mathcal{P}_{\mathcal{AC}}$, again his disclosure policy $\mathcal{P}_{\mathcal{D}}$, the sets $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{N}}$ (step 11).
6. In the case of computed missing foreign credentials $\mathcal{C}_{\mathcal{M}}$, Bob transforms $\mathcal{C}_{\mathcal{M}}$ into requests for credentials and awaits until receives all responses. At this point Bob acts as a client, requesting Alice the set of credentials $\mathcal{C}_{\mathcal{M}}$. Alice runs the same protocol with swapped roles.
7. When Bob receives all responses he restarts the loop and consults the `iAccessDecision` algorithm for a new decision.
8. When a final decision of grant or deny is taken, Bob checks and resumes all suspended threads awaiting on the current negotiation and then returns the decision back to Alice.

The distribution and issuance of credentials are second order problems which can be solved in a number of ways not interesting in this setting. As indicated in the figure, we use the keyword parfor for representing that the body of the loop is run each time in a parallel thread. Thus, each missing credential is requested independently from the requests for the others. At that point of the protocol, it is important that each of the finished threads updates presented and declined sets of foreign credentials properly without interfering with the other threads. We point out that each thread updates the requested credential as declined after a certain session time expires.

Example 3. Figure 9 shows an example of Alice's and Bob's interactions using the negotiation protocol on both sides. The policies for access to resources and access to sensitive credentials are in notations like in Yu *et al.* [9] where the Alice's local credentials are marked with subscript "A" and Bob's with "B," respectively. Bob's access policy $\mathcal{P}_{\mathcal{AR}}$ says that access to resource r_1 is granted if $\{C_{A1}, C_{A2}\}$ or, alternatively, if $\{C_{A1}, C_{A3}\}$ are presented by Alice. To get access to r_2 Alice should satisfy the requirements for access to r_1 and present C_{A4}.

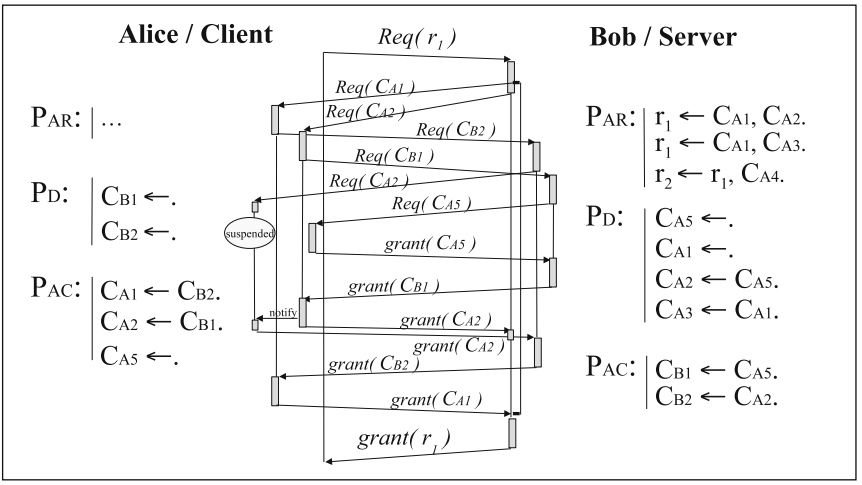

Fig. 9. Example of interoperability of the negotiation protocol.

We read Bob's disclosure policy as to disclose the need for a credential C_{A2} there should be already disclosed a credential C_{A5}, which by default is always disclosable. Credential C_{A1} is always disclosable on demand, but in contrast, the need for C_{A4} is never disclosed but expected from $\mathcal{P}_{\mathcal{AR}}$ when r_2 is requested. It is an example of a hidden credential that must be pushed.

Bob's policy $\mathcal{P}_{\mathcal{AC}}$ states that access to Bob's C_{B1} is granted if Alice has presented C_{A5} and that access to Bob's C_{B2} if Alice's C_{A2}.

7. STEPWISE DISCLOSURE OF ACCESS RIGHTS

The intuition here is that Bob may *not* want to disclose the missing foreign credentials all at once to Alice but, instead, he may want to ask Alice first some less sensitive credentials assuring him that Alice is trustworthy enough to disclose her more sensitive credentials and so on until all of the missing ones are disclosed.[4] To address this issue we extend the protocol in Section 6 with an algorithm for stepwise disclosure of missing credentials. The basic intuition is that the logical policy structure itself tells us which credentials must be disclosed to obtain the information that other credentials are missing. So, we simply need to extract this information automatically. We perform a step-by-step evaluation on the policy structure. For that purpose we use the one-step deduction (Definition 5)

[4]The stepwise approach may require a client to provide credentials that are not directly related to a specific resource but needed for a fine-grained disclosure control.

```
iAccessNegotiation(r, C_P){ ...}
AskCredentials(C_M)
 1: while C_M ⋢ (C_P ∪ C_N) do
 2:     C_M1 = StepwiseDisclosure(C_M);
 3:     if C_M1 == ∅ or C_M1 == null then return;
 4:     parfor each c∈C_M do
 5:       response = invoke iAccessNegotiation(c, ∅)@Opponent;
 6:       if response == grant then C_P = C_P ∪{ c } else C_N = C_N ∪{ c };
 7:     done
 8:     while C_M1 ⋢ (C_P ∪ C_N) do wait(timeout);
 9: done
StepwiseDisclosure(C_M)
 1: C_D1 = {c | C_P| = P_D^1 c} \ (C_P ∪ C_N);
 2: P_D1 = {ĉ ← B | c←B∈P_D }∪{ c← ĉ | c←B∈P_D and c ∉(C_D1 ∪ C_N)};
 3: Q= { q← ∧_{c∈C_M} c};
 4: result = abduction(q, C_D1, P_D1 ∪ C_P ∪Q) ;
 5: return result;
```

Fig. 10. The stepwise negotiation protocol.

over the disclosure policy $\mathcal{P}_\mathcal{D}$ to determine the next set of potentially disclosable credentials.

Essentially, the protocol replaces the `AskCredentials` function with a new version of it using the stepwise disclosure algorithm, see Fig. 10. With its new version the `AskCredentials` function (Fig. 10) takes as input the set of missing credentials $\mathcal{C}_\mathcal{M}$ (as the old one) and internally loads the policy for disclosure control $\mathcal{P}_\mathcal{D}$ ($\mathcal{C}_\mathcal{M}$ was computed from). In a nutshell, the algorithm requests a client all missing credentials supplied in the input, but with the difference of stepwise awaiting for each of the computed steps by the `StepwiseDisclosure` algorithm. In other words, when a next step of missing credentials is computed (step 2) the algorithm awaits until the client responds to all current requests. Again here the client's profile is updated on each request/response to facilitate other threads' access decisions. Then the check in step 3 (for $\mathcal{C}_{\mathcal{M}1}$) comprises two cases: either the set of presented foreign credentials $\mathcal{C}_\mathcal{P}$ has been updated (indirectly) by other running threads such that now $\mathcal{C}_\mathcal{M}$ is satisfied and there is no next step or the client has declined some credentials that stop his way further to $\mathcal{C}_\mathcal{M}$.

The task of the `StepwiseDisclosure` is to determine at each interaction step exactly the relevant credentials needed to reach at the end the set $\mathcal{C}_\mathcal{M}$. For doing so, in step 1, we compute the set of disclosable foreign credentials $\mathcal{C}_{\mathcal{D}1}$ by

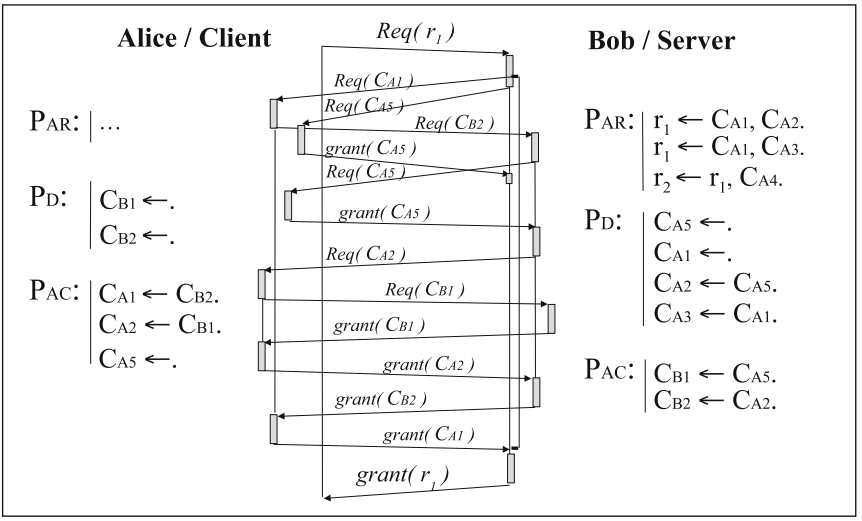

Fig. 11. Example of stepwise negotiation of credentials.

one-step deduction over $\mathcal{P}_\mathcal{D}$ and according to $\mathcal{C}_\mathcal{P}$. Out of those, we extract only the minimal set of credentials that is actually necessary to derive $\mathcal{C}_\mathcal{M}$. To this extent, we modify policy $\mathcal{P}_\mathcal{D}$ by adding a new atom q that can be derived if all (and only) credentials in $\mathcal{C}_\mathcal{M}$ are derived (refer to step 3). Additionally, we also change syntactically the structure of $\mathcal{P}_\mathcal{D}$ rules so that relevant credentials in $\mathcal{C}_{\mathcal{D}1}$ must be abduced and can no longer be derived from chaining other rules in $\mathcal{P}_\mathcal{D}$ (step 2).

We do that by changing a rule of the from $c \leftarrow c_1, \ldots, c_n$ into a pair of rules $\hat{c} \leftarrow c_1, \ldots, c_n$ and $c \leftarrow \hat{c}$, where $\hat{c}$ is a new symbol. The informal meaning of the first rule is that c is disclosable if all c_i are. So, we now say that the need for the fictitious $\hat{c}$ is disclosable if the need for all c_i is disclosable and that the need for the credential c is disclosable if the need for $\hat{c}$ is. Then if we remove $c \leftarrow \hat{c}$, for all $c \in \mathcal{C}_{\mathcal{D}1} \cup \mathcal{C}_\mathcal{N}$ there will be no rule to infer that c is disclosable so we must abduce it as a primitive atom (if it is actually needed to derive q). For that purpose we specify in the abduction input $\mathcal{C}_{\mathcal{D}1}$ as the set of hypotheses so that all $c \in \mathcal{C}_{\mathcal{D}1}$ are potentially abducible but none of the $\mathcal{C}_\mathcal{N}$ are. Thus, for deriving the next step of foreign missing credentials we invoke the `abduction` engine with input $\langle q, \mathcal{C}_{\mathcal{D}1}, \mathcal{P}_{\mathcal{D}1} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{Q} \rangle$ and refer the reader to Definition 6 for the properties of the abduction computation.

Figure 11 shows an example of how the stepwise disclosure algorithm works, following the same scenario as in Example 3.

Notice that parties might end in a deadlock when a negotiator requests credential A in order to give B, while the other party needs B to give A. This is likely

possible as the protocol allows multiple sub-negotiations in different threads and blocks when the requested credential is already in negotiation.

Practically, we have solved the problem with a timeout: if we do not get an answer from a partner in a reasonable time we assume that the credential is not available. Notice that this is also necessary to avoid honest partners of being victims of malicious clients who may simply "forget" to answer and thus saturate the opponent's memory with many unfinished threads.

8. CONCLUSIONS AND FUTURE WORK

In this paper we proposed a policy-based negotiation scheme for access rights establishment. The key idea is that in an autonomic communication a client and a server have an autonomic view on their access polices and privacy requirements and, as so, they need a way to reason and negotiate the requirements to access a resource.

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

The paper contribution is in the way one can bootstrap from the basic access control algorithm a comprehensive negotiation scheme. The scheme proposes a model and a protocol that allow entities to bilaterally negotiate access control requirements. The protocol runs on client and server sides so that they understand each other and automatically interoperate until an agreement is reached or denied.

We enriched the framework over the existing policy-based negotiation approaches [7–9] by introducing two different policies: one for protecting resources and one for protecting sensitive credentials. The distinction extends the negotiation framework on a wider set of policy languages with respect to the existing approaches.

We extended the negotiation protocol with a stepwise disclosure algorithm that provides a fine-grained privacy control over the disclosure of missing credentials. The algorithm enforces stepwise sequences of credentials which a foreigner should follow in order to get access to a resource.

One of the advantages in our approach is that we do not pose any restrictions on partner's policies because the basic computations of deduction and abduction do not require any specific policy structure. Thus, we can support hierarchical policies (indeed arbitrary policies for which reasoning services exist), non-monotone policy, and by using one-step deduction also fine granularity control on the information disclosure.

An important issue behind the stepwise approach is that it may introduce new hurdles during a negotiation (and thus failures to access resources). Negotiations may begin with a partial policy disclosure but may fail eventually because the

client cannot fulfill all the requirements to reveal the remaining part of the policy. In other words, a client might have a credential *A* needed to get a service, but without a credential *B*, possibly not needed for any service but only to disclose the need for *A*, he will be denied access. Whether an obstacle or a feature it is application dependent and as so it can be turned on or off on demand.

Future work is in the direction of proving what guarantees the protocol offers in terms of interoperability (completeness and correctness) when applied to other negotiation schemes such as TrustBuilder by Yu *et al*. [9]. We believe that this is an important step toward building a secure open computing environment.

ACKNOWLEDGMENTS

This work was partly supported by the projects: 2003-S116-00018 PAT-MOSTRO, 016004 IST-FP6-FET-IP-SENSORIA, 27587 IST-FP6-IP-SERENITY, 038978 EU-MarieCurie-EIF-iAccess, 034744 EU-INFSO-IST ONE, 034824 EU-INFSO-IST OPAALS.

REFERENCES

1. M. Sloman and E. Lupu, Policy specification for programmable networks. In *Proc. of the 1st Intl. Working Conference on Active Networks*, pp. 73–84. Springer-Verlag, 1999.
2. L. Lymberopoulos, E. Lupu, and M. Sloman, An adaptive policy based framework for network services management, *Journal of Network and Systems Management*, Vol. 11, No. 3, pp. 277–303, 2003, Plenum Press.
3. M. Smirnov, Rule-based systems security model. In *Proc. of MMM-ACNS*, pp. 135–146, 2003, Springer-Verlag Press.
4. S. De Capitani di Vimercati and P. Samarati, Access control: Policies, models, and mechanism. In R. Focardi and F. Gorrieri (eds.), *Foundations of Security Analysis and Design*, Vol. 2171 of LNCS. Springer-Verlag Press, 2001.
5. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, Role-based access control models, *IEEE Computer*, Vol. 39, No. 2, pp. 38–47, 1996.
6. SPKI, SPKI certificate theory, 1999. IETF RFC 2693. Available from http://www.ietf.org/rfc/rfc2693.txt.
7. N. Li and J. C. Mitchell, RT: A role-based trust-management framework. In *Proc. of DISCEX III Conf.*, pp. 201–212, 2003, IEEE press.
8. P. Bonatti and P. Samarati, A unified framework for regulating access and information release on the web, *Journal of Computer Security*, Vol. 10, No. 3, PP. 241–272, 2002.
9. T. Yu, M. Winslett, and K. E. Seamons, Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation, *ACM Transactions on Information and System Security*, Vol. 6, No. 1, pp. 1–42, 2003.
10. T. Yu and M. Winslett, A unified scheme for resource protection in automated trust negotiation. In *Proc. IEEE Symposium on Security and Privacy*, pp. 110–122, May 2003 IEEE press.
11. H. Koshutanski and F. Massacci, Interactive access control for Web Services. In *Proc. of IFIP Information Security Conference*, pp. 151–166, 2004, Kluwer.
12. H. Koshutanski and F. Massacci, Interactive credential negotiation for stateful business processes. In *Proc. of iTrust Conference*, pp. 257–273, 2005, Springer-Verlag Press.

13. H. Koshutanski and F. Massacci, Abduction and deduction in logic programming for access control for autonomic systems. Tech. Report, DIT-05-053, University of Trento, July 2005. http://eprints.biblio.unitn.it/archive/00000821/01/053.pdf.
14. K. Apt, Logic programming. In J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*. Elsevier, 1990.
15. M. Gelfond and V. Lifschitz, The stable model semantics for logic programming. In *Proc. of the 5th International Conference on Logic Programming*, pp. 1070–1080, 1988, MIT-Press.

Hristo Koshutanski has a PhD in Computer Science from the University of Trento, Italy. He holds the SATIN-EDRF award for doctoral research in 2005. In 2006 he won a EU Marie Curie Fellowship. He is a Research Associate at the University of Malaga, Spain. His research interests include distributed system security, trust management, access control models and authorization policies.

Fabio Massacci is full professor in Informatics at the University of Trento, Italy, and guest scientist at SINTEF, Norway. He was a visiting researcher at IRIT—Toulose, France, assistant professor at the Univ. of Siena, post doctoral fellow and got a PhD at the Univ. of Roma I "La Sapienza" in 1998. His main research interests are Computer Security, Formal Verification, and Requirements Engineering. He published a number of articles on international conferences and journals and is responsible for a number of EU and national research grants. He is rectors's delegate for ICT procurements and services. He is member of ACM and IEEE.