Interactive Credential Negotiation for Stateful Business Processes*

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. Business Processes for Web Services are the new paradigm for lightweight enterprise integration. They cross organizational boundaries, are provided by entities that see each other just as business partners, and require access control mechanisms based on trust management. Stateful Business Processes, enforcing separation of duties or service limitations based on past or current usage, pose additional research challenges. Clients, which may not know the right set of credentials to supply to each partner, may end up in dead-ends and servers should help them find out what must be revoked and what missing is that grant access to a particular resource.

We propose a logical framework and an interactive algorithm based on negotiation of credentials for access control that works for Stateful Business Processes. We show that our algorithm is sound (no grant is given to unauthorized clients), complete (authorized clients get grant) and resistant against DoS attempt.

1 Introduction

Business Processes (BPs) for Web Services (WS) are the new buzzword for e-commerce integration. BPs allow for a lightweight integration of partners' services and the establishment of virtual enterprises on the Web. To support this effort a number of standards have emerged: SOAP and WSDL for basic functionalities, BPEL4WS and ebXML for complex business processes.

Business Processes can be seen as workflows distributed among independent partners where all communication is channeled by the invocation of web services. Since each "task" of the workflow is offered as a web service that can be activated by anyone credentials must be used to enforce access control.

In this paper we discuss our system for reasoning about access control for Stateful BPs for WS. The basic intuition is that partners, offering web services in a BP, do not know a priori what credentials clients may need to present nor

* This work is partially funded by the IST programme of the EU Commission FET under the IST-2001-37004 WASP project and by the FIRB programme of MIUR under the RBNE0195K5 ASTRO Project and RBAU01P5SS Project.

P. Herrmann et al. (Eds.): iTrust 2005, LNCS 3477, pp. 256–272, 2005.

clients know exactly which services they want, as the BP may take different paths. So, we need an interactive process in which a client starts a business process and partners evaluate client's current credentials to determine whether they are sufficient to unlock a resource or something is missing. If missing is found then they communicate it back to the client which, in turn, may decline some of the requested credentials and a new path must be sought.

If Business Processes are stateless this can be accomplished using either trust negotiation by Yu et al. [1], Bonatti and Samarati's framework for access to Web Services [2] or our own framework for interactive access control [3] with various degree of automation, flexibility and restriction on policies. For example Yu et al. reasures monotone policies.

If the access control to a BP is stateful (i.e. access decision can change depending on past interactions or past presented credentials) this is no longer possible. For example separation of duties means that we cannot escalate privileges by supplying more credentials. Past requests or services may deny access to future services as in Bertino et al. [4] centralized access control model for workflows.

So, we need to find a way for BP partners to find a solution assuming they only know their policies. Further, it does not make sense for a BP to ask all potentially useful credentials (too demanding and privacy intruding for clients) nor such option is practical, considering that partners may prefer to ask for some credentials directly the clients rather than making them publicity available.

1.1 Our Contribution

In this paper we give an algorithm for full-fledged access control for stateful BP and show that it is correct, complete and resistant against malicious clients.

The intuition behind an interactive access control algorithm is the following.

- Initially a client submits a set of credentials and a service request.
- Then the algorithm checks whether the request is granted by the access policy according to the client's set of credentials.
- If the check fails the algorithm computes all credentials disclosable from the disclosure policy according to the presented credentials.
- After that, using abductive reasoning, the algorithm finds a (minimal) solution set of missing credentials that unlocks the desired resource and preserves consistency.
- If such a set cannot be found, the algorithm performs a recovery step in which it runs the abductive reasoning again to find a (minimal) set of excessing credentials that ban the client to get a solution for the resource.
- Once a solution (missing or excessing credentials) is found it is communicated back to the client so that he can provide the missing credentials and revoke the excessing ones.

We note that in contrast to intra-enterprise workflow systems [4], a partner offering a service has no way to assign to a client the right set of credentials which would be consisted with his future requests (because the partner cannot

assign to or prohibit the client future tasks). So, we must have some roll-back procedure by which, if the user has sent "wrong" credentials, he can revoke them.

Following is a short introduction to the basic framework and reasoning services. Section 3 introduces the interactive access control algorithm for stateful BP. Next, Section 4 extends the interactive algorithm to cope with malicious clients and shows that the extended version is resistant to Denial of Service attacks. Theoretical part of the stateful framework together with the theorems for soundness and completeness is presented in Section 5. Then, Section 6 introduces the global management and initialization of session profiles in the framework. Finally, Section 7 looks through the related work and concludes the paper.

2 Basic Framework

This section shortly introduces the basic model in our framework and we refer the reader to [3] for more details. The formal model is based on normal logic programs under the stable model semantics [5]. We have predicates for requests, credentials, assignments of users to roles and of roles to services, see Figure 1. They are self explanatory, except for role dominance: a role dominates another if it has more privileges. We have constants for user's identifiers, denoted by $\mathsf{User}:U$, for roles, denoted by $\mathsf{Role}:R$, and one for services, denoted by $\mathsf{Service}:S$.

Policies are written as normal logic programs. These are sets of *rules*:

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

$\mathsf{Role}:R_i \succ \mathsf{Role}:R_j$ when role $\mathsf{Role}:R_i$ dominates role $\mathsf{Role}:R_j$.
$\mathsf{Role}:R_i \succ_{\mathsf{Service}S} \mathsf{Role}:R_j$ when for service $\mathsf{Service}:S$, role $\mathsf{Role}:R_i$ dominates role $\mathsf{Role}:R_j$.
$\mathsf{assign}\,(P, \mathsf{Service}:S)$ when an access to the service $\mathsf{Service}:S$ 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 $\mathsf{Role}:R$.
$\mathsf{credentialTask}\,(\mathsf{User}:U, \mathsf{Service}:S)$ when $\mathsf{User}:U$ has the right to access $\mathsf{Service}:S$.

(b) Predicates for Credentials

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

(b) Predicates for System's History and State

Fig. 1. Predicates used in the model

where A, B_i and C_i are (possibly ground) predicates among those shown in Figure 1. 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*. In our framework, we also need *constraints* that are rules with an empty head.

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

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 situations in which B_i are true and all C_j are false (those situations are not acceptable).

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.*

This reasoning service is used in most logical formalizations [6]: if the request r is a consequence of the policy ($\mathcal{P}$) and the credentials ($\mathcal{C}$) (i.e. $\mathcal{P} \cup \mathcal{C} \models r$), then access is granted otherwise it is denied.

A number of works have deemed such blunt denials unsatisfactory and therefore it has been proposed by Bonatti and Samarati [2] and Yu et al. [1] to send back to the client some of the rules that are necessary to gain additional access. Figure 2 shows the essence of the approaches. In their work it is revised to allow for the flow of rules and information to users.

1. verify that the request is a logical consequence of the credentials, namely $\mathcal{P} \cup \mathcal{C} \models r$,
2. if the check succeeds then grant access else
   (a) select some rule $r \leftarrow p \in \texttt{PartialEvaluation}(\mathcal{P} \cup \mathcal{C})$,
   (b) if r exists then send the rule back to the client else deny access.

Fig. 2. Disclosable Access Control

The systems proposed in [2, 1] are flat, i.e. in p the client will find all missing credentials to continue the process until r is granted. In many cases this is neither sufficient nor desirable. 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 structure is used, these systems would not be complete.

Here, a partner must be able to infer the causes of a failed request and to ask a client the missing credentials. The corresponding logical process is no longer

deduction but it is *abduction*. So we must have co-existence of deduction (for deciding access and disclosure of information) and abduction (for explaining failed requests).

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.

How we bootstrap from the two basic reasoning services a comprehensive interactive access control algorithm for BPs is the subject of the next section.

3 Interactive Access Control for Stateful Systems

Stateful systems are systems where the status of the current state depends on the status of the system in past conditions, i.e. access decision can change depending on past interactions or past presented credentials.

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.

To execute a service of the fragment of a partner, the user will submit a set of *presented credentials* $\mathcal{C}_p$, a set of *revoked credentials* $\mathcal{C}_r$ and a *service request* r. We assume that $\mathcal{C}_p$ and $\mathcal{C}_r$ are disjoint. We also need to keep a memory of past credentials submitted by a user. This is the role of $\mathcal{C}_{\mathcal{P}}$, the set of *active credentials* that have been presented by the client in past requests to other services within the domain of a partner.

In many workflow authorization schemes, the policy alone is not sufficient to make an access control decision and thus we need to identify a *history of execution* $\mathcal{H}$ of services under the control of a partner. It keeps track on what has been done by the system and what is the current status of it. For instance a branch manager of a bank clearing a cheque cannot be the same member of staff who has emitted the cheque [4–pag.67]. If we had no memory of past credentials then it would be impossible to enforce any security policy for separation of duties on the application workflow.

Once a client makes a service request, the authorization mechanism starts a session in which the client iterates with the system until a final decision of *grant* or *deny* is taken. In the same session context, we keep a set of *declined credentials* $\mathcal{C}_{\mathcal{N}}$, a set of *missing credentials* $\mathcal{C}_{\mathcal{M}}$ and a set of *excessing credentials* $\mathcal{C}_{\mathcal{E}}$. The set $\mathcal{C}_{\mathcal{N}}$ consists of credentials that the client has declined to present to the system during an authorization session. The sets $\mathcal{C}_{\mathcal{M}}$ and $\mathcal{C}_{\mathcal{E}}$ keep information from the output of the last interaction. Once the session is started, the algorithm loads the policies for access and disclosure control $\mathcal{P}_{\mathcal{A}}$

and $\mathcal{P}_\mathcal{D}$ together with the two sets: the history of execution $\mathcal{H}$ and the client's active credentials $\mathcal{C}_\mathcal{P}$.

Our interactive access control solution for stateful services and applications is shown in Figure 3. The logical explanation of the algorithm is the following. The algorithm's input consists of client's sets of currently presented credentials $\mathcal{C}_p$, revoked ones $\mathcal{C}_r$ and the service request r. When a client requests a specific service the authorization mechanism creates a new session and initializes to empty set the variables $\mathcal{C}_\mathcal{N}$, $\mathcal{C}_\mathcal{M}$ and $\mathcal{C}_\mathcal{E}$.

Then, the set of active credentials $\mathcal{C}_\mathcal{P}$ is updated by removing the revoked ones $\mathcal{C}_r$ from it and then adding the newly presented credentials $\mathcal{C}_p$ (ref. step 1). The declined credentials $\mathcal{C}_\mathcal{N}$ are updated by credentials the client was asked in the previous interaction minus the ones that he has currently presented. Step 3 prepares the two sets $\mathcal{C}_\mathcal{M}$ and $\mathcal{C}_\mathcal{E}$ for the interaction output.

Next, the algorithm checks whether the request r is granted by $\mathcal{P}_\mathcal{A}$ and $\mathcal{C}_\mathcal{P}$ (step 4). If the check fails then in step 5a the algorithm computes all credentials disclosable from $\mathcal{P}_\mathcal{D}$ and $\mathcal{C}_\mathcal{P}$ and from the resulting set removes all already declined and presented credentials. In this way we avoid dead loops of asking something already declined or presented. Step 5b computes (using abduction reasoning) a (minimal) solution for r. Up to this point the algorithm is essentially our interactive access control algorithm for stateless WS described in [3].

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

1. update $\mathcal{C}_\mathcal{P} = (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_r) \cup \mathcal{C}_p$,
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. Set up $\mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{E} = \emptyset$,
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 $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \not\models \bot$,
5. if the check succeeds then return ***grant*** else
   (a) compute the set of *disclosable credentials* $\mathcal{C}_\mathcal{D} = \{c \mid \mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{P} \models c\} \setminus (\mathcal{C}_\mathcal{N} \cup \mathcal{C}_\mathcal{P})$,
   (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***($\mathcal{C}_\mathcal{E}$)> and iterate else
      i. for every $c \in \mathcal{C}_\mathcal{P}$ introduce a new credential $\hat{c}$ in the language,
      ii. use abduction to find a minimal set of *missing credentials* $\mathcal{C}_\mathcal{M} \subseteq \{\hat{c} \mid c \in \mathcal{C}_\mathcal{P}\} \cup \mathcal{C}_\mathcal{D}$ such that
         – $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \{c \leftarrow \text{not } \hat{c}. \mid c \in \mathcal{C}_\mathcal{P}\} \cup \mathcal{C}_\mathcal{M} \models r$,
         – $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \{c \leftarrow \text{not } \hat{c}. \mid c \in \mathcal{C}_\mathcal{P}\} \cup \mathcal{C}_\mathcal{M} \not\models \bot$,
      iii. if no set $\mathcal{C}_\mathcal{M}$ exists then return ***deny*** else
         A. compute $\mathcal{C}_\mathcal{E} = \{c \mid \hat{c} \in \mathcal{C}_\mathcal{M}\}$ and $\mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{M} \cap \mathcal{C}_\mathcal{D}$,
         B. return <***ask***($\mathcal{C}_\mathcal{M}$),***revoke***($\mathcal{C}_\mathcal{E}$)> and iterate.

Fig. 3. Interactive Access Control Algorithm for Stateful Services

If in step 5c no $\mathcal{C}_\mathcal{M}$ was found then we come to the part of the algorithm devoted to stateful systems. The motivation here is that if a solution for r cannot be found in $\mathcal{C}_\mathcal{D}$ it means that

- either the client *does not have enough privileges* to get the disclosure of more missing credentials so that the abduction can find a solution
- or in the client's set of credentials $\mathcal{C}_\mathcal{P}$ there is *something "wrong"* that bans the client to get any solution, i.e. it makes $\mathcal{P}_\mathcal{A}$ inconsistent.

In the first case there is nothing you can do and we should just quietly deny access. In the second case we could have a possibility for recovery.

So, following the second case, in steps 5(c)i and 5(c)ii, we use abduction over the set of disclosable and active credentials $\mathcal{C}_\mathcal{D} \cup \mathcal{C}_\mathcal{P}$ searching for a possible solution $\mathcal{C}_\mathcal{M}$ that unlocks r and preserves consistency in $\mathcal{P}_\mathcal{A} \cup \mathcal{H}$. If a solution for r is found it clearly indicates that this solution could not be found in step 5b because of the existence of "wrong" credentials in $\mathcal{C}_\mathcal{P}$ that makes $\mathcal{P}_\mathcal{A}$ inconsistent. In this case we compute the set of excessing credentials $\mathcal{C}_\mathcal{E}$ as the set difference $\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{M}$. Here we separate the definitely good (consistent) solution $\mathcal{C}_\mathcal{M}$ from the rest in $\mathcal{C}_\mathcal{P}$.

Notice that steps 5(c)i–5(c)iii could be simplified by simply setting $\mathcal{C}_\mathcal{E} = \mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{M} = \emptyset$, i.e. asking the client to revoke everything and restart from scratch. We believe that this is hardly practical. We want to have a more precise control on the revokable credentials i.e. of being able to compute a minimal set of revokable and missing credentials. To do so we introduce for each credential $c \in \mathcal{C}_\mathcal{P}$ a new symbol for it $\hat{c}$ in the model, step 5(c)i in Figure 3. Then after obtaining the set of new symbols $\{\hat{c} \mid c \in \mathcal{C}_\mathcal{P}\}$ we generate a set of rules $\{c \leftarrow \text{not } \hat{c}. \mid c \in \mathcal{C}_\mathcal{P}\}$. The trick here is that negating all credentials in $\mathcal{C}_\mathcal{P}$ using the newly introduced symbols and running abduction reasoning over the set union of $\{\hat{c} \mid c \in \mathcal{C}_\mathcal{P}\} \cup \mathcal{C}_\mathcal{D}$ (step 5(c)ii) allows us to find a *minimal* solution $\mathcal{C}_\mathcal{M}$ for r that itself indicates what should be revoked and what should be asked from the client.

Let us consider the set $\{c \leftarrow \text{not } \hat{c}. \mid c \in \mathcal{C}_\mathcal{P}\}$. Since we attach this set to $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P}$ so it follows that all credentials in $\mathcal{C}_\mathcal{P}$ will be deduced again except those that the corresponding new symbol appears in $\mathcal{C}_\mathcal{M}$. So, all new-symbol credentials appearing in $\mathcal{C}_\mathcal{M}$, computed in step 7(c)ii, will be treated such that the absence of their respective credentials in $\mathcal{C}_\mathcal{P}$ allows the abduction reasoning to find a solution set for r.

Remark 1 (Multiple Activations and Revocations of Credentials). In an interactive access control process the algorithm may ask the client to present credentials that he has revoked (was explicitly asked for that) in previous interactions or ask the client to revoke credentials that the same has activated.

Since we do not know what solution a client has for a particular resource so in the presence of alternatives in the access policy the system may choose the "wrong" one[1] such that later on when the right alternative is chosen it may require the

[1] Here we call "wrong" alternative a solution set that the client does not have it.

revocation/activation of credentials that were already activated/revoked by the client in the interactions with the"wrong" one.

Example 1. Abstracting from a specific meaning let us have the following scenario. A client with a set of available credentials $\{C_A, C_B, C_C\}$ wants to access a service r. The client's set of active credentials (already presented to the system) is $\mathcal{C}_\mathcal{P} = \{C_C\}$ and history $\mathcal{H} = \emptyset$. The policies for access and disclosure control are shown below.

$$\mathcal{P}_\mathcal{A} \left| \begin{array}{l} r \leftarrow C_A, C_B. \\ r \leftarrow C_C, C_D. \\ \leftarrow C_A, C_C. \end{array} \right. \qquad \mathcal{P}_\mathcal{D} \left| \begin{array}{l} C_A \leftarrow . \\ C_B \leftarrow . \\ C_C \leftarrow . \\ C_D \leftarrow . \end{array} \right.$$

Now, let suppose that the client initially requests service r with set of presented credentials $\mathcal{C}_p = \{C_A\}$. According to the algorithm in Figure 3 the check in step 4 will fail, then in step 5b abduction will not find any solution because the policy is inconsistent with the client's set of credentials and so the algorithm will reach step 5(c)ii. The output of this step, considering the minimality criterion subset containment, is:

- Abduction output: $\left\{\hat{C}_A, C_D\right\}$, algorithm result: ***ask(***$\{C_D\}$***)***, ***revoke(***$\{C_A\}$***)***
- Abduction output: $\left\{\hat{C}_C, C_B\right\}$, algorithm result: ***ask(***$\{C_B\}$***)***, ***revoke(***$\{C_C\}$***)***

Since we do not know what solution the client has so we must choose one of the two outcomes listed above.

If we are lucky and choose the solution <***ask(***$\{C_B\}$***)***, ***revoke(***$\{C_C\}$***)***> then on the next interaction since the client has in possession $\{C_A, C_B, C_C\}$ the same presents C_B, revokes C_C and gets ***grant*** r in step 5 in the next interaction.

In the other case, if we choose the solution <***ask(***$\{C_D\}$***)***, ***revoke(***$\{C_A\}$***)***> then on the next interaction the client will revoke C_A but will decline to present C_D, simply because he does not have it. Then the check in step 4 will not succeed because $\mathcal{C}_\mathcal{P} = \{C_C\}$ does not contain enough credentials to unlock r. Following that, the abduction reasoning is step 5b will not find a solution because in $\mathcal{C}_\mathcal{P}$ there is a credentials C_C that is inconsistent with the only solution available in $\mathcal{C}_\mathcal{D} = \{C_A, C_B\}$.

Running again abduction reasoning with minimality according subset containment the only solution found is $\{\hat{C}_C, C_A, C_B\}$ and the respective outcome of the algorithm is <***ask(***$\{C_A, C_B\}$***)***, ***revoke(***$\{C_C\}$***)***>. Essentially, we ask the client to restart from scratch.

On the next interaction since the client has in possession $\{C_A, C_B, C_C\}$ so he revokes C_C, presents $\{C_A, C_B\}$ and in step 5 gets *grant* r.

4 Coping with Malicious Clients

We need to improve the algorithm in Figure 3 to protect the server against Denial of Service (DoS) attacks. To do so we consider the client as an entity

that can manipulate the system only via its input sets of credentials $\mathcal{C}_p$ and $\mathcal{C}_r$. Particularly, he may present in his input set $\mathcal{C}_p$ credentials, which he has revoked in past interactions with the system, without been explicitly asked for it and, respectively, may revoke credentials in $\mathcal{C}_r$, which he has activated and presented to the system in past interactions, again without been asked for it. In both cases it would bring the system in a previous state (by letting it compute the same solution again and again). A malicious client could thus waste server's time forever.

The new algorithm is shown in Figure 4. In step 1, a new data set is introduced, the set of *revoked credentials* $\mathcal{C}_\mathcal{R}$, which accumulates all credentials revoked by a client in an interaction session for a particular service r. So, step 1 updates the set of revoked credentials by removing from $\mathcal{C}_\mathcal{R}$ the set of missing credentials $\mathcal{C}_\mathcal{M}$, asked in the last interaction, and adding to the resulting set the newly revoked credentials $\mathcal{C}_r$. The motivation for the set difference $\mathcal{C}_\mathcal{R} \setminus \mathcal{C}_\mathcal{M}$ is that whatever the client presents from $\mathcal{C}_\mathcal{M}$ it is dropped from $\mathcal{C}_\mathcal{R}$ because it is not any more revoked and whatever it is not presented in $\mathcal{C}_\mathcal{M}$ but is dropped from $\mathcal{C}_\mathcal{R}$ is added to the set of declined credentials $\mathcal{C}_\mathcal{N}$. In this case revoked and declined credentials are kept disjoint, i.e. $\mathcal{C}_\mathcal{R} \cap \mathcal{C}_\mathcal{N} = \emptyset$.

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

1. update $\mathcal{C}_\mathcal{R} = (\mathcal{C}_\mathcal{R} \setminus \mathcal{C}_\mathcal{M}) \cup (\mathcal{C}_r \cap \mathcal{C}_\mathcal{E})$,
2. update $\mathcal{C}_\mathcal{P} = (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{R}) \cup (\mathcal{C}_p \setminus \mathcal{C}_\mathcal{R}) \cup (\mathcal{C}_p \cap \mathcal{C}_\mathcal{M}) \cup (\mathcal{C}_p \cap \mathcal{C}_\mathcal{N})$,
3. update $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{N} \cup (\mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p)$,
4. update $\mathcal{C}_\mathcal{U} = \mathcal{C}_\mathcal{U} \cup (\mathcal{C}_\mathcal{E} \setminus \mathcal{C}_r)$,
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***($\mathcal{C}_\mathcal{E}$)> and iterate else
      i. for every $c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})$ introduce a new credential $\hat{c}$ in the language,
      ii. use abduction to find a minimal set of *missing credentials* $\mathcal{C}_\mathcal{M} \subseteq \{\hat{c} \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{D}$ such that
         - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup (\mathcal{C}_\mathcal{P} \cap \mathcal{C}_\mathcal{U}) \cup \{c \leftarrow \text{not } \hat{c}. \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{M} \models r$,
         - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup (\mathcal{C}_\mathcal{P} \cap \mathcal{C}_\mathcal{U}) \cup \{c \leftarrow \text{not } \hat{c}. \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{M} \not\models \bot$,
      iii. if no set $\mathcal{C}_\mathcal{M}$ exists then return ***deny*** else
         A. compute $\mathcal{C}_\mathcal{E} = \{c \mid \hat{c} \in \mathcal{C}_\mathcal{M}\}$ and $\mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{M} \cap \mathcal{C}_\mathcal{D}$,
         B. return <***ask***($\mathcal{C}_\mathcal{M}$),***revoke***($\mathcal{C}_\mathcal{E}$)> and iterate.

Fig. 4. Extended Access Control Algorithm

Extending further step 1, to prevent situations of revoking credentials not supposed to be revoked by the client, we update $\mathcal{C}_{\mathcal{R}}$ by adding only those credentials from $\mathcal{C}_r$ that the client was explicitly asked in the previous interaction, i.e. adding only $\mathcal{C}_r \cap \mathcal{C}_{\mathcal{E}}$.

Step 2 updates the client set of active credentials by removing from $\mathcal{C}_{\mathcal{P}}$ the set of all revoked credentials and adding to it the set of newly activated credentials $\mathcal{C}_p$. First we use the set difference $\mathcal{C}_{\mathcal{P}} \setminus \mathcal{C}_{\mathcal{R}}$ to remove all revoked credentials and second, expanding the set of active credentials, we add from currently presented credentials $\mathcal{C}_p$ only the credentials that have not been revoked before – $\mathcal{C}_p \setminus \mathcal{C}_{\mathcal{R}}$ – and we add also those credentials in $\mathcal{C}_p$ that the system has asked the client – $\mathcal{C}_p \cap \mathcal{C}_{\mathcal{M}}$ – or the client has declined to present in past interactions but it is presenting now – $\mathcal{C}_p \cap \mathcal{C}_{\mathcal{N}}$.

In other words, step 2 allows the client to activate credentials among those:

- that the system has asked him to present in the last interaction or
- that he has denied to present in past interactions or
- brand new credentials[2] that the client has not supplied to the system at the time of interacting.

Then, in step 4, we introduce a new data set $\mathcal{C}_{\mathcal{U}}$. The role of $\mathcal{C}_{\mathcal{U}}$ is analogous of that for $\mathcal{C}_{\mathcal{N}}$ and serves as a data store for those credentials that a client has declined to revoke in an interaction session. $\mathcal{C}_{\mathcal{U}}$ is updated by adding to it the set difference of excessing credentials the client was asked in the last interaction minus the ones currently revoked. Similarly, once a client refuses to revoke a credential the same credential will not be considered in a possible output again.

We note that a client at any time can present credentials that he has declined to present in previous interactions, although he will never be asked for them again. Here, a client is not allowed (the system will not consider) to revoke credentials that he has refused to revoke before without been asked for it. The last requirement is mainly because the revocation of credentials is usually a cumbersome process and once the client refuses to revoke a credential then it is unlikely to expect him to do it later in a negotiation process.

The last key-point of the extended algorithm is in step 7(c)ii. Here we run the abduction reasoning over the set $\{\hat{c} \mid c \in (\mathcal{C}_{\mathcal{P}} \setminus \mathcal{C}_{\mathcal{U}})\} \cup \mathcal{C}_{\mathcal{D}}$. The set difference $\mathcal{C}_{\mathcal{P}} \setminus \mathcal{C}_{\mathcal{U}}$ comes from the fact that we do not want to ask the client to revoke credentials that he has already refused to revoke. In this way we rule out those models where the client already denied to comply to. We also note that the two conditions in step 7(c)ii are analogous with their respective ones in Figure 3 because whatever we drop from $\mathcal{C}_{\mathcal{P}} \setminus \mathcal{C}_{\mathcal{U}}$ we add it by the intersection of $\mathcal{C}_{\mathcal{P}} \cap \mathcal{C}_{\mathcal{U}}$.

Now on, wherever we refer to the access control algorithm we refer to its extended model shown in Figure 4.

[2] Excluding the revoked credentials since the system is aware of them.

5 Correctness and Completeness

This section presents the theoretical part of the stateful framework together with theorems for soundness and completeness. We refer the reader to [7] for details on the proofs of the theorems.

At first we introduce some preliminary definitions.

Definition 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 $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{S} \not\models \bot$).*

Definition 4 (Completeness). *If a client has a solution for a request r then he always gets* grant r.

Definition 5 (Soundness). *If a client gets* grant r *then he has a solution for r.*

In the following we assume that the sets of missing and excessing credentials are disjoint, i.e. $\mathcal{C}_\mathcal{M} \cap \mathcal{C}_\mathcal{E} = \emptyset$, otherwise the server will reject the answer outright. Note that this is true for the interactive algorithm in Figure 4. We also assume that at any time in an interaction process the sets of currently presented and revoked credentials are disjoint, i.e. $\mathcal{C}_p \cap \mathcal{C}_r = \emptyset$.

Definition 6 (Powerful and Compliant Client). *A* powerful and compliant client *is a client that whenever receives $<$**ask**$(\mathcal{C}_\mathcal{M})$,**revoke**$(\mathcal{C}_\mathcal{E})>$ returns $<\mathcal{C}_\mathcal{M},\mathcal{C}_\mathcal{E}>$, i.e. activates all $c \in \mathcal{C}_\mathcal{M}$ and revokes any $c \in \mathcal{C}_\mathcal{E}$.*

Definition 7 (Cooperative and Compliant Client). *A client with ability to manage (obtain or revoke) a set of credentials $\mathcal{C}$ is a* cooperative and compliant client *if whenever receives $<$**ask**$(\mathcal{C}_\mathcal{M})$,**revoke**$(\mathcal{C}_\mathcal{E})>$ returns $<\mathcal{C}_\mathcal{M} \cap \mathcal{C}, \mathcal{C}_\mathcal{E} \cap \mathcal{C}>$, i.e. activates all $c \in (\mathcal{C}_\mathcal{M} \cap \mathcal{C})$ and revokes any $c \in (\mathcal{C}_\mathcal{E} \cap \mathcal{C})$.*

Definition 8 (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.*

Definition 9 (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 then $\mathcal{C}_\mathcal{S}$ is disclosable by $\mathcal{P}_\mathcal{D}$, i.e. $\forall c \in \mathcal{C}_\mathcal{S}$, $\mathcal{P}_\mathcal{D} \models c$.*

Theorem 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 with the algorithm in Fig. 4 then he has a solution set $\mathcal{C}_\mathcal{S}$ that unlocks r according to $\mathcal{P}_\mathcal{A}$.*

Theorem 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 in Fig. 4 always terminates.*

We use the idea of well-founded tuples ordering so that at each interaction we associate the tuple $<\mathcal{C}_\mathcal{N}, \mathcal{C}_\mathcal{U}, \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R}>$. Then we show that if the algorithm does not return grant or deny (it terminates) then the sets in the tuple always increase from one interaction to the next so that the maximal number of interactions is bounded by the credentials occurring in the access policy.

5.1 Completeness

Definition 10 (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 then any superset $\mathcal{C}' \supset \mathcal{C}$ is also a solution set for r according to P.*

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. $P \cup \mathcal{C}' \not\models r$*

Theorem 3 (Completeness for a Monotonic Access Policy). *Let $\mathcal{P}_\mathcal{A}$ be a* monotonic *access policy, $\mathcal{P}_\mathcal{D}$ be a monotonic disclosure policy and r a request. If $\mathcal{P}_\mathcal{A}$ and $\mathcal{P}_\mathcal{D}$ guarantee fair access and interaction then a powerful or cooperative client always gets* grant *r with the algorithm in Fig. 4.*

Of course the whole business of the stateful systems requires non-monotonic policies, so this result is not enough. Here we relax the policy access $\mathcal{P}_\mathcal{A}$ from the assumption of monotonicity and consider it as an arbitrary non-monotonic policy. So, from now on we assume that $\mathcal{P}_\mathcal{A}$ is *non-monotonic* and $\mathcal{P}_\mathcal{D}$ *monotonic* unless explicitly specified otherwise.

In the same context, without loss of generality, we assume that whenever a client initially requests a service he submits $\mathcal{C}_p = \mathcal{C}_r = \emptyset$ and if hidden credentials $\mathcal{C}_\mathcal{H}$ needed then $\mathcal{C}_p = \mathcal{C}_\mathcal{H}$ and $\mathcal{C}_r = \emptyset$.

Theorem 4 (Completeness for a Powerful and Compliant Client). *Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy, $\mathcal{H}$ be history of executions and r a request. If $\mathcal{P}_\mathcal{A} \cup \mathcal{H}$[3] and $\mathcal{P}_\mathcal{D}$ guarantee fair access and interaction then a powerful and compliant client always gets grant r with the algorithm in Fig. 4.*

Here the properties for fair access and interaction guarantee that it exists a solution $\mathcal{C}_\mathcal{S}$ for r that is disclosable by the disclosure policy and so the abduction reasoning

[3] It is important to pose the property of fair access over $\mathcal{P}_\mathcal{A} \cup \mathcal{H}$ because it guarantees fairness wrt other (possibly) environment constraints like limited number of executions on a service, limited number of users accessing a service and so on.

potentially can find it in step 7b in Figure 4. If it fails then in the next step 7(c)ii because of the existence of such solution and since it is contained in the set of disclosable credentials $\mathcal{C}_\mathcal{D}$ follows that abduction at least finds this solution together with the set of excessing credentials which is returned back to the client. Then because the client is a powerful one he presents and revokes what he was asked and on the next interaction gets grant.

Theorem 5 (Completeness for a Cooperative and Compliant Client). *Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy, $\mathcal{H}$ be history of executions and r a request. If $\mathcal{P}_\mathcal{A} \cup \mathcal{H}$ and $\mathcal{P}_\mathcal{D}$ guarantee fair access and interaction then if a cooperative and compliant 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 with the algorithm in Fig. 4.*

We prove it in two parts. First part showing that if a client does not get grant he gets $<\text{ask}(\mathcal{C}_\mathcal{M}), \text{revoke}(\mathcal{C}_\mathcal{E})>$, i.e. he never gets deny. Second, we use Theorem 2 (for termination) and conclude that in a bounded number of interaction a cooperative client always gets grant.

6 Initialization and Service Management

Figure 5 gives the intuition of possible management of services wrt the access control decision process and its relevant data sets. The algorithm shown in Figure 5 works like web servers. A main web server listens to service requests and whenever a request is detected the server runs the algorithm in a new thread and initializes the global variables $\mathcal{H}$ and $\mathcal{C}_\mathcal{P}$ to empty sets. Of course here it should distinguish between the very first initialization and any further loading of those data sets simply because we do not want to loose any data from past interactions. For further loadings we keep a profile for each client with the respective data set $\mathcal{C}_\mathcal{P}$ so that when the client accesses again a service we just load $\mathcal{C}_\mathcal{P}$.

Global Vars: $\mathcal{C}_\mathcal{P}$, $\mathcal{H}$;
Initially: $\mathcal{C}_\mathcal{P} = \emptyset$ and $\mathcal{H} = \emptyset$;

`ServiceRequest`$(r, \mathcal{C}_p, \mathcal{C}_r)\{$ // starts a new thread

1. $result_{access}$ = `InteractiveAccessControl`$(r, \mathcal{C}_p, \mathcal{C}_r)$;
2. `Update`$(\mathcal{H}, result_{access}, r)$;
3. if $result_{access} == grant$ then
4. $\quad result_{service}$ = `InvokeService`(r);
5. $\quad$ `Update`$(\mathcal{H}, result_{service}, r)$;
6. endif

}

Fig. 5. Global Initialization and Management of Services

Algorithm output	Status Predicates
grant	grant (User : U, Service : S, Number : N), running (User : U, Service : S, Number : N)
deny	deny (User : U, Service : S, Number : N)

Service Execution	Status Predicates
accomplished	success (User : U, Service : S, Number : N)
failed	abort (User : U, Service : S, Number : N)

Both $\mathcal{C}_\mathcal{P}$ and $\mathcal{H}$ are local to a service provider and are managed and initialized independently. The history of execution $\mathcal{H}$ is set up to the empty set at the start when a particular business process is started[4]. Even a business partner may decide to have for each running business process separate histories $\mathcal{H}$. To this extend we assume that $\mathcal{H}$ is mapped to those business process(es) that are relevant to the authorization logic and is released when those processes complete their executions.

Another session profile is the set of active user's access rights $\mathcal{C}_\mathcal{P}$ available to the partner's application domain. Each session is associated with a single user and each user is associated with a *single* session. This session profile is created[5] when the user for the first time requests a service under the partner's domain. In contrast to the history profile, the set of user's access rights is valid until a certain time slot expires. Even more, it is valid across multiple runnings of business processes within the entire scope of the partner's domain and it eventual deactivation depends on the partner's authorization logic.

The third session profile kept in the model is for the service level negotiation. Here, each session is associated with a single user but each user is associated with *one or more* negotiation sessions. Whenever a user requests a service (ref. ServiceRequest(...) in Figure 5) it is created a session within which the interactive access control algorithm is running. Once the session is created the user interacts with the system until a final decision of grant or deny is taken. Within these interactions a user (de)activates some subset of roles ($\subseteq \mathcal{C}_\mathcal{P}$) that he or she is assigned.

In comparison with RBAC model, the service level agreement session corresponds to the *user_sessions(u:USERS)* function as introduced by Ferraiolo et al. [8], which is the mapping of a user u onto a set of active sessions. The user session of active rights corresponds to *avail_session_perms(user_sessions(u:USERS))* introduced in [8]. Where *avail_session_perms(s:SESSIONS)* returns the permissions available to a user in a session. We note that the user's session of active access rights in our framework extends Ferraiolo et al. [8] because in $\mathcal{C}_\mathcal{P}$ one can also find access credentials from already concluded service level sessions but with still valid expiration dates.

To keep the history of execution $\mathcal{H}$ up-to-date, after each interaction step appropriate predicates should be added indicating what has been done by the system.

[4] It entirely depends on the provider's business logic and whenever an application business process is started we refer to it as an initial point to set up $\mathcal{H} = \emptyset$.

[5] The set $\mathcal{C}_\mathcal{P}$ is strictly time-dependent and must be periodically cleared up from already expired credentials.

This is done by the function `Update` shown in Figure 5. The table below summarizes the possible updates of $\mathcal{H}$.

The temporal evolution of the access rights wrt the history of execution $\mathcal{H}$ can be complex because even the most simple constraint on executed actions may block a request. Indeed, the set of requests that must be grantable by the policy may change with the services that we have used. As intuitively expected, we may have access to less services if we have limitations on their usage.For instance the following constraint specifies that the service *reviewSellBids* cannot be executed more than three times in a workflow session:

$$\leftarrow \mathsf{assign}\,(\mathsf{User}:U, \mathsf{Service}:reviewSellBids),$$
$$4 \leq \{N.\,\mathsf{success}\,(\mathsf{User}:U, \mathsf{Service}:reviewSellBids, \mathsf{Number}:N)\}\,.$$

7 Related Work and Conclusions

Access control for business processes 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 [9], role based access control on the web [10, 11], tasks [12] and DRM [9], 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 work on trust negotiations [13, 1] focuses on communication and infrastructure and assumes 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, 14, 15, 16] do not treat the problem of inferring missing credentials from failed requests.

Also the proposal by Bonatti and Samarati [2] 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 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 and then will evaluate these credentials according its release policy.

The other key proposal on trust negotiation by Yu et al. [1], offers a dual view w.r.t Bonatti and Samarati [2]. 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 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 [1–page 21], it is impossible to access resources if some of the needed credentials cannot be disclosed at some point.

In this paper we have presented a framework for interactive access control for stateful systems. The work is the continuation of [3] and goes beyond the model for stateless services. With the extended framework a service provider is able to reason of not only finding the missing credentials that compliment client's access rights to unlock the resource but also to resolve conflicts occurring in its security policy. The last thing is in the context of reasoning about the excessing credentials, among the client's ones, that make the provider's policy inconsistent. Even more, the proposed access control algorithm breaks off the monotonic policy paradigm so that now a service provider can control access to its resources written in an arbitrary non-monotonic policy language.

The work models two types of clients, a powerful and a cooperative, and together with the definitions for fair access and interaction they are stated theorems for soundness and completeness for the two clients.

Finally, we refer the reader to [17, 7] for an architectural approach and implementation of the interactive access control framework and to [18] for an extension of it that copes with bilateral negotiation of credentials.

References

1. Yu, T., Winslett, M., 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** (2003) 1–42
2. Bonatti, P., Samarati, P.: A unified framework for regulating access and information release on the web. Journal of Computer Security **10** (2002) 241–272
3. Koshutanski, H., Massacci, F.: Interactive access control for Web Services. In: Proceedings of the 19th IFIP Information Security Conference (SEC 2004), Toulouse, France, Kluwer Press (2004) 151–166
4. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. ACM Transactions on Information and System Security (TISSEC) **2** (1999) 65–104
5. Apt, K.: Logic programming. In van Leeuwen, J., ed.: Handbook of Theoretical Computer Science. Elsevier (1990)
6. 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)
7. Koshutanski, H., Massacci, F.: Interactive access control for stateful web services business processes. Technical Report DIT-05-002, Department of Information and Communication Technology, University of Trento (2005)
8. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM TISSEC **4** (2001) 224–274
9. Park, J., Sandhu, R.: Towards usage control models: beyond traditional access control. In: Seventh ACM SACMAT, ACM Press (2002) 57–64
10. Giuri, L.: Role-based access control on the web. ACM Transactions on Information and System Security (TISSEC) **4** (2001) 37–71
11. Park, J.S., Sandhu, R.: RBAC on the Web by smart certificates. In: Proceedings of the fourth ACM workshop on Role-based access control, ACM Press (1999) 1–9
12. Joshi, J.B.D., Aref, W.G., Ghafoor, A., Spafford, E.H.: Security models for web-based applications. Communications of the ACM **44** (2001) 38–44

13. Roscheisen, M., Winograd, T.: A communication agreement framework for access/action control. In: Proceedings of the Symposium on Security and Privacy, IEEE Press (1996) 154–163
14. Li, N., Grosof, B.N., Feigenbaum, J.: Delegation logic: A logic-based approach to distributed authorization. ACM Transactions on Information and System Security (TISSEC) **6** (2003) 128–171
15. Jajodia, S., Samarati, P., Subrahmanian, V.S., Bertino, E.: A unified framework for enforcing multiple access control policies. In: Proceedings of the 1997 ACM SIGMOD conference on Management of data, ACM Press (1997) 474–485
16. Wijesekera, D., Jajodia, S.: Policy algebras for access control the predicate case. In: Proceedings of the 9th ACM conference on Computer and Communications Security, ACM Press (2002) 171–180
17. Koshutanski, H., Massacci, F.: An access control framework for business processes for Web services. In: Proceedings of the 2003 ACM workshop on XML security, Fairfax, VA, ACM Press (2003) 15–24
18. Koshutanski, H., Massacci, F.: An interactive trust management and negotiation scheme. In: Proceedings of the 2nd International Workshop on Formal Aspects in Security and Trust (FAST), Toulouse, France, Kluwer Press (2004) 139–152