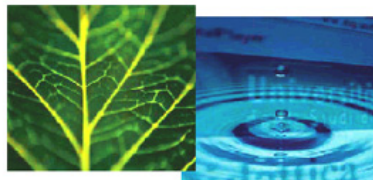


PhD Dissertation



UNIVERSITY OF TRENTO

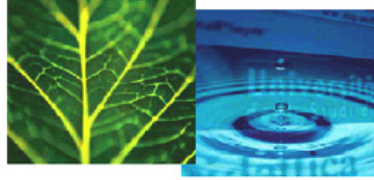
INTERNATIONAL GRADUATE SCHOOL IN INFORMATION AND COMMUNICATION
TECHNOLOGIES

MARCH 2005

Interactive Access Control for Autonomic Systems

Hristo Koshutanski

PhD Dissertation



UNIVERSITY OF TRENTO

INTERNATIONAL GRADUATE SCHOOL IN INFORMATION AND COMMUNICATION
TECHNOLOGIES

MARCH 2005

Hristo Koshutanski

Interactive Access Control for Autonomic Systems

Advisor:
Fabio Massacci

Committee Members:
Piero Bonatti
Pierangela Samarati
Mikhail Smirnov

External Reviewers:
William Winsborough

AUTHOR'S ADDRESS:

Hristo Koshutanski

Dipartimento di Informatica e Telecomunicazioni

Università degli Studi di Trento

Via Sommarive 14, I-38050 Povo di Trento, Italy

E-MAIL: hristo.koshutanski@dit.unitn.it

WWW: <http://dit.unitn.it/~hristo>

Abstract

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

To solve this problem we propose to use a novel *interactive access control* model: servers should be able to get back to clients asking for missing or excessing credentials, whereas the clients may supply or decline the requested credentials. The process iterates until a final decision of grant or deny is taken.

This proposal is grounded in a formal model on policy-based access control. It identifies the automated reasoning services of deduction, abduction and consistency checking that characterize the problem. Based on them, it proposes a comprehensive access control framework for stateless and stateful autonomic systems. The framework describes two interactive access control algorithms for stateless and stateful services and shows their correctness and completeness.

On top of the interactive access control, the dissertation presents an interactive trust negotiation model. The negotiation model 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 automatically negotiate the requirements to access a service.

The dissertation ends by presenting the implementation of the interactive access control model, called *iAccess*. X.509 standard is used as a main certificate infrastructure for describing participants' identities and access rights. SAML specification is used for having standard semantics for authorization statements among participants. SOAP protocol is used for making the access engine Web Services compatible when interacting with other applications. DLV system is used as a back-end engine for the basic functionalities of deduction and abduction.

Keywords

Automated Reasoning, Deduction, Abduction, Adaptive and Self-healing Access Control Systems, Conflict Detection and Resolution, Credential-based Trust Management, Pervasive Computing, Logics for Trust Management, Trust Negotiation.

to my family Dimitriyka and Stoyanka

Contents

Acknowledgements	vii
1 An Intuitive Introduction to Interactive Access Control	1
1.1 Introduction	1
1.2 A Running Example	2
1.3 Introducing Interactive Access Control	3
1.4 The Contribution of this Dissertation	8
1.5 Dissertation Outline	9
2 Survey on Distributed Access Control	11
2.1 Authorization Requirements	11
2.2 Access Control Architectures	12
2.2.1 Single Policy-based Access Control	13
2.2.2 Multi-Policy Based Access Control	16
2.3 Access Control Systems and Security Requirements	18
3 The Logical Model	19
3.1 Syntax and Semantics	19
3.2 Automated Reasoning in Answer Set Programming	21
3.3 Formalization of the Running Example	22
4 Interactive Access Control	25
4.1 Stateless Autonomic Systems	25
4.2 Stateful Autonomic Systems	27
4.3 Coping with Malicious Clients	31
4.4 Stateful Session Data Management	33
4.5 Technical Guarantees	34
5 Theoretical Results	37
5.1 Stateless Framework	37
5.2 Completeness for Powerful and Cooperative Clients	38
5.3 Stateful Framework	41
5.4 Completeness for Powerful and Cooperative Compliant Clients	44
6 Trust Negotiation	49
6.1 The Trust Negotiation Model	49
6.2 Access vs Negotiation Control	50
6.3 The Negotiation Protocol	50
6.4 Piecewise Disclosure	54

7	System Architecture	57
7.1	A Primer on Web Services and Business Processes	57
7.2	The Architecture and Components	59
7.3	Integration with Web Services Technologies	60
7.4	<i>iAccess</i> Prototype	62
7.5	Integration with Automated Reasoning Tool DLV	65
8	Conclusions	67
8.1	Open Problems and Future Work	68

Acknowledgements

Can a PhD degree be evaluated by only 150 pages of scientific work? From the very beginning of my doctorate degree I have been taught, guided and mentored of how to do research, how to approach new ideas and how to mature in scientific environment. All of these I am deeply indebted and grateful to my advisor Fabio Massacci. Indeed, he has given me much more than just to simply thank him for my graduation. He is a professor not only in academia but in life which makes him invaluable advisor and friend. Without his ‘special style supervision’ I would not have the same personality which goes beyond a 150-page dissertation.

I would also like to thank the dean of the Faculty of Mathematics and Informatics at Plovdiv University “Paisii Hilendarski”, Prof. Dimitar Mekerov, for giving me great support and guidance during my student years. He gave me the initial incitement for academic life and the motivation to begin my doctorate degree.

I express my gratitude to the members of the committee for their constructive and useful comments. Individually, to Piero Bonatti, Pierangela Samarati and Mikhail Smirnov for being internal referees and to William Winsborough acting as an external referee.

I am eternally grateful to my mother Dimitriyka Traeva for the moral strength and encouragement she gave me during the difficult moments I had.

Last but not least, I want to thank the University of Trento and particularly the department of Information and Communication Technology for the PhD position I was granted, which makes this work possible. I would like to acknowledge the financial sponsorship of the University of Trento and the Ministero Affari Esteri for my scholarship and the research grants FIRB-ASTRO, EU-IST WASP and E-NEXT SATIN award which supported travels for conferences and workshops to present the work discussed in this dissertation.

Trento, March 2005.

List of Figures

1.1	Joint Hierarchy Model	2
1.2	Traditional Access Control	4
1.3	Disclosable Access Control	4
1.4	Basic Idea of Interactive Access Control	5
1.5	Interactive Access Control with Controlled Disclosure	6
1.6	Interactive Access Control for Stateful Systems	8
2.1	Woo and Lam Framework [55]	13
2.2	Akenti Architecture [51]	14
2.3	PERMIS Architecture [15]	14
2.4	Adage System [64]	15
2.5	Praesidium Auth. Server [52]	15
2.6	OASIS Architecture [5]	16
2.7	Cross-domain Authorization [4]	17
2.8	RAD Architecture [8]	17
3.1	Planet-Lab Access and Disclosure Control Policies	23
4.1	Interactive Access Control Algorithm for Stateless Services	26
4.2	Interactive Access Control Algorithm for Stateful Services	28
4.3	DoS Resistant Interactive Access Control Algorithm for Stateful Services	31
4.4	Global Initialization and Service Management	33
6.1	The Trust Negotiation Protocol	52
6.2	Example of Interoperability of the Negotiation Protocol	53
6.3	The Piecewise Negotiation Protocol	54
6.4	Example of Piecewise Negotiation of Credentials	55
7.1	Web Services Technology Stack	57
7.2	Example of a BPEL4WS Process	58
7.3	System Architecture	59
7.4	Overall View of the System Architecture and Integration	61
7.5	X.509 Identity and Attribute Certificates Structure	62
7.6	<i>i</i> Access Architecture	64
7.7	Implementation of the Basic Functionalities of Deduction and Abduction	66

List of Tables

2.1	Summary of the Access Control Systems and the Authorization Requirements	18
3.1	Predicates Used in the Logical Model	19
3.2	Status Predicates Used in the Stateful Model	20

Chapter 1

An Intuitive Introduction to Interactive Access Control

This chapter introduces the starting point of interactive access control by comparing it with the related work. It presents a running example to make the discussion more concrete and outlines the contribution of the dissertation.

1.1 Introduction

Recent advances of Internet technologies and globalization of peer-to-peer communications offer for organizations and individuals an open environment for rapid and dynamic resource integration. In such an environment federations of heterogeneous systems are formed with no central authority and no unified security infrastructure. Considering this level of openness each server is responsible for the management and enforcement of its own security policies with a high degree of autonomy.

Controlling access to services is a key aspect of networking and the last few years have seen the domination of policy-based access control. Indeed, the paradigm is broader than simple access control and one may speak of *policy-based network self-management* (See [48, 40] or the IEEE Policy Workshop series for examples). The intuition is that actions of nodes controlling access to services are automatically derived from policies. The nodes look at events, requested actions and credentials presented to them, evaluate the policy rules according to those new facts and derive the actions [48, 49]. Policies can be “simple” **iptables** configuration rules for Linux firewalls¹ or complex logical policies expressed in languages such as Ponder [17] or a combination of policies across heterogeneous systems as in OASIS XACML framework [59].

Dynamic coalitions and autonomic communication add new challenges: a truly autonomic network is born when nodes are no longer within the boundary of a single enterprise, which could deploy its policies on each and every node and guarantee interoperability. An autonomic network is characterized by properties of self-awareness, self-management and self-configuration of its constituent nodes. In an autonomic network, nodes are like partners that offer services and lightly integrate their efforts into one (hopefully coherent) network. This cross enterprise scenario poses novel security challenges with aspects of both trust management systems and workflow security.

From trust management systems [54, 19, 36] it takes the credential-based view. Since access to network services is offered by autonomic nodes on their own and to potentially unknown clients, the decision to grant or deny access can only be made on the basis of credentials sent by a client.

From workflow access control systems (e.g., [3, 7, 22, 25]) we borrow all classical problems such as dynamic assignment of roles to users, dynamic separation of duties and assignment of permissions

¹See <http://www.netfilter.org/>.

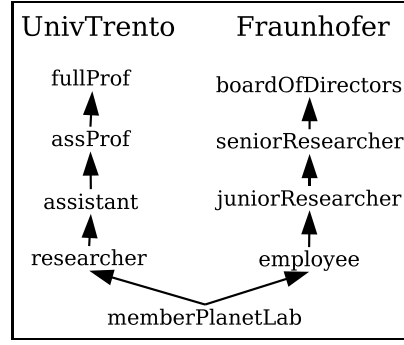


Figure 1.1: Joint Hierarchy Model

to users according to the least privilege principles.

In an autonomic communication scenario a client might have all the necessary credentials to access a service but may simply not know it. Equally, it is unrealistic to assume that servers will publish their security policies on the web so that clients can do a policy combination and evaluation themselves. So, it should be possible for a server to ask a client, on the fly, for additional credentials whereas the client may disclose or decline to provide them. Next, the server re-evaluates the client's request, considering the newly submitted credentials and computes an access decision. The process iterates between the server and the client until a final decision of grant or deny is taken. We call this modality *interactive access control*.

Part of these challenges can be solved by using policy-based self-management of networks but not all of them. Indeed, if we abstract away the details on the policy implementation, one can observe that the only automated reasoning service that is actually used by policy-based self-management approaches is *deduction*: given a policy and a set of additional facts find out all consequences (actions or obligations) from the policy according to the facts. We simply look whether granting the request can be deduced from the policy and the current facts. Policies could be different [6, 36, 12, 7] but the kernel reasoning service is the same.

Access control for autonomic communications needs another, less-known, automated reasoning service, taken from AI domain, called *abduction* [47]. Loosely speaking, we could say that abduction is deduction in reverse: given a policy and a request to access a network service we want to know what are the credentials (facts) that would grant access. Logically, we want to know whether there is a (possibly minimal) set of facts that added to the policy would entail (deduce) the request.

If we look again at our intuitive description of the interactive access control, it is immediate to realize that abduction is the core service needed by the policy-based autonomic servers to reason for missing credentials.

We can also use abduction on a client side so that whenever a client is requested for missing credentials he can perform evaluation on his policy and counter-request the server for some evidences in order to establish enough confidence (trust) to disclose the originally requested credentials.

1.2 A Running Example

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

Members of the two labs are classified in a hierarchy that is shown in Figure 1.1. The figure shows the joint hierarchy model of the roles at both institutions, Fraunhofer and University of Trento. The partial order of roles is indicated by arcs, where higher the role in the hierarchy, more powerful it is. A role *dominates* another role if it is higher in the hierarchy and there is a direct path between them.

The access policy of the Planet-Lab network specifies that:

- **disk** access is allowed for any request coming from the two institutions,
- **run** access is allowed for any request coming from specific machines at the two institutions or any request coming from the two institutions accompanied with a membership certificate,
- **configure** access is allowed to anybody that has **run** access to the network resources and is at least researcher at University of Trento or junior researcher at Fraunhofer institute. Further extending access rights, **configure** access is also granted to associate professors or senior researchers with the requirement of accessing the Planet-Lab network from the respective country domains of Italy or Germany. The least restrictive access is granted to full professors or members of board of directors obliging them to provide the appropriate credential attesting their positions.

Let us have the scenario where Alice is a senior researcher at Fraunhofer and daily she needs to get **run** access to resources at Planet-Lab network. So, whenever she is at her office and she wants to execute some services she sends her employee certificate to the system. According to the access policy, **run** access is granted to Alice because as an employee she is a member of the Planet-Lab hierarchy model (ref. Figure 1.1).

Now, examine the case in which Alice wants to have access to the system from his home place (deciding to work at home) presenting her employee certificate assuming that it is potentially enough to get **run** access to certain services. But, according to the policy rules the system should deny the request because **run** access requests coming from domains different than University of Trento or Fraunhofer institute are allowed only to associate professors or senior researchers or higher role-positions.

So, the natural question is, "is it the behavior we want from the system?" Shall we leave Alice with only "access denied" decision and being idle for the whole day simply because she did not know or just has forgotten that access to the system outside Fraunhofer needs another certificate?

An answer like "sorry, we also need a credential for being at least a senior researcher" would be more than welcomed by most employees. At the same time, the server wants to be sure to ask this additional credential only to employees.

A full formalization of the example is given in §3.3.

1.3 Introducing Interactive Access Control

In this section we introduce, step by step, the novel contribution of interactive access control by "evolving" the existing access control frameworks.

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

Figure 1.2 shows the "traditional" access control decision process. Whether the decision process uses RBAC [45], SDSI/SPKI [50], RT [37] or any other trust management framework it is immaterial

at this stage: they can be captured by suitably defining \mathcal{P}_A , \mathcal{C}_p and the entailment operator (\models). This approach is the cornerstone of most logical formalizations [18]: if the request r is a consequence of the policy and the credentials then access is granted otherwise it is denied.

-
1. check whether \mathcal{P}_A and \mathcal{C}_p entail r ,
 2. if the check succeeds then **grant** access
 3. else **deny** access.
-

Figure 1.2: Traditional Access Control

A number of works has deemed such blunt denials unsatisfactory. Bonatti and Samarati [12] and Yu et al. [63] proposed to send back to clients some of the rules that are necessary to gain additional access. Figure 1.3 shows the essence of the approaches.

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

Figure 1.3: Disclosable Access Control

Both works have limitations because they impose severe 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).

The first and foremost limitation is that both approaches requires policies to be flat: a policy protecting a resource must contain all credentials needed to allow 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.

Constraints that would make policy reasoning non-monotone (such as separation of duties) are also ruled out as they 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 structure 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 [62], which becomes a limitation when information disclosure plays crucial role.

The work by Yu and Winslett [62] 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. These limitations are due to a traditional view point: the only reasoning service one needs for access view point is *deduction*, i.e., check that the request follows from the policy and the presented credentials.

Intuition 1 We claim that we need another, less known, reasoning service *abduction*, i.e., check which 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* in Figure 1.4.

-
1. check whether \mathcal{P}_A and \mathcal{C}_p entail r ,
 2. if the check succeeds then **grant** access
 3. else
 - (a) compute a set \mathcal{C}_M such that:
 - \mathcal{P}_A together with \mathcal{C}_p and \mathcal{C}_M entail r , and
 - \mathcal{P}_A together with \mathcal{C}_p and \mathcal{C}_M preserve consistency.
 - (b) if \mathcal{C}_M exists then ask the client for \mathcal{C}_M and iterate
 - (c) else **deny** access.
-

Figure 1.4: Basic Idea of Interactive Access Control

The "compute a set \mathcal{C}_M such that ..." (step 3a) 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}_A and, therefore, we need a way to define how to control the disclosure of such a set.

As we have already noted, Yu and Winslett addressed partly this issue by protecting policies within the access policy itself. However, this is not really satisfactory as it does not decouple the decision about access from the decision about disclosure. Consider Yu and Winslett's own example:

Example 1 (Yu and Winslett [62, page 4])

[Access Scenario] McKinley clinic makes its patient records available for online access. Let r be Alice's record. To gain access to r a requester must either present Alice's patient ID for McKinley clinic ($C_{AliceID}$), or present a California social worker license (C_{CSWL}) and a release-of-information credential (C_{RoI}) issued to the requester by Alice.

[Disclosure Scenario] Alice wants to keep the latter constraint inaccessible to strangers. However, employees of McKinley clinic ($C_{McKinleyEmployee}$) should be allowed to see the contents of the policy. \square

We note that the disclosure requirement for $C_{McKinleyEmployee}$ cannot be captured via the service accessibility scheme by Bonatti and Samarati [12] and refer to [62] for details. We also point

out that having $C_{McKinleyEmployee}$ does not allow access to r but rather is used to unlock more information on how to access r .

So, from a standpoint of a good engineering practice a structured approach of separate access and disclosure policies is better than a flat (merged policy) approach because the criteria behind and 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.

Intuition 2 We claim that 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}_D . The policy for disclosure control is used to decide credentials whose need can be potentially disclosed to a client. In other words, \mathcal{P}_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}_D defines which credentials among those occurring in \mathcal{P}_A are disclosable so, if needed, can be demanded from the requestor.

The relevant approach wrt the disclosure policy \mathcal{P}_D , is the one by Yu and Winslett [62]. It postulates that policies for protecting resources should be themselves treated (protected) 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, 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}_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.

Thus, we can give a new refined algorithm for *interactive access control with controlled disclosure* shown in Figure 1.5.

-
1. check whether \mathcal{P}_A and \mathcal{C}_p entail r ,
 2. if the check succeeds then **grant** access
 3. else
 - (a) compute the set of disclosable credentials \mathcal{C}_D entailed by \mathcal{P}_D and \mathcal{C}_p ,
 - (b) compute a set \mathcal{C}_M out of the disclosable ones ($\mathcal{C}_M \subseteq \mathcal{C}_D$) such that
 - \mathcal{P}_A together with \mathcal{C}_p and \mathcal{C}_M entail r , and
 - \mathcal{P}_A together with \mathcal{C}_p and \mathcal{C}_M preserve consistency.
 - (c) if \mathcal{C}_M exists then ask the client for \mathcal{C}_M and iterate
 - (d) else **deny** access.
-

Figure 1.5: Interactive Access Control with Controlled Disclosure

Example 2 Example 1 formalized as two logic programs:

$$\begin{array}{l|l} \mathcal{P}_D: & \mathcal{P}_A: \\ \hline C_{AliceID} \cdot & r \leftarrow C_{AliceID} \cdot \\ C_{CSWL} \leftarrow C_{McKinleyEmployee} \cdot & r \leftarrow C_{CSWL}, C_{RoI} \cdot \\ C_{RoI} \leftarrow C_{McKinleyEmployee} \cdot & \end{array}$$

The disclosure control policy is read as the disclosure of Alice's ID is not protected and potentially

released to anybody. The need for disclosing the credentials for California social worker license C_{CSWL} and release-of-information C_{RoI} is released only to users that have already presented their McKinley employee certificates $C_{McKinleyEmployee}$.

The access policy specifies that access to r is granted either to Alice or to California social workers that have a release-of-information credential issued by Alice. \square

There are still tricky questions to be answered such as:

- How do we know that the algorithm terminates? In other words, can a client waste the server's time forever?
- How do we know that a cooperative client, starting with a wrong set of credentials, can actually arrive to a grant? For example, can we assure that the server will not keep asking Alice for a UNITN full professor credential which she does not have, while never asking for a FOKUS senior researcher credential, which she has?

We will show how to fix the details of the algorithm in Chapter 4 so that all answers are positive.

This is enough to cover stateless systems.

We still have a major challenge ahead: how do we cope with stateful systems? Stateful systems are systems where the access decisions change depending on past interactions or past presented credentials. Such systems can easily become inconsistent wrt the client's set of presented credentials mainly because access policies may forbid the presentation of credential if another currently active credential has been presented in the past.

Past requests or services may deny access to future services as in Bertino et al. [7] centralized access control model for workflows. Separation of duties means that we cannot extend privileges by supplying more credentials. For instance a branch manager of a bank clearing a cheque cannot be the same member of staff who has emitted the cheque [7, page 67]. If we have no memory of past credentials then it is impossible to enforce any security policy for separation of duties on the application workflow. The problems that could cause a process to get stuck are the following:

- the request may be inconsistent with some role, action or event from the client in the past;
- the new set of credential may be inconsistent with requirements such as separation of duties;

Intuition 3 We claim that in the stateful systems' domain we not only need to reason on what missing credentials allow access but also on what are *excessing* (conflicting) credentials that make the policy state inconsistent. We need a procedure by which if a user has exceeded his privileges he has the chance to revoke them.

The basic intuitive algorithm for *interactive access control for stateful systems* is shown in Figure 1.6. Steps 1 to 3d are essentially the basic interactive access control algorithm (ref. Figure 1.5) with controlled disclosure. The part for stateful systems comes when we are not able to find a set of missing credentials among the disclosable ones (step 3d).

In this case there are two reasons which may cause the abduction failure when computing \mathcal{C}_M . The first one could be that in \mathcal{C}_D there are not enough disclosed credentials to unlock r , case in which we should deny access (step 3(d)iii), or there might be credentials in the client's set of presented credentials \mathcal{C}_p that make the policy state inconsistent so that any solution among the disclosable credentials cannot be found by the abduction.

The latter reason motivates step 3(d)i. In this step, first, we want to find a set of conflicting credentials \mathcal{C}_E , called *excessing*, among the presented ones \mathcal{C}_p such that removing them from \mathcal{C}_p preserves the access policy consistent, step 3(d)iA. Second, on top of the not conflicting credentials it must exist a solution set that entails the service request, step 3(d)iB. The second requirement assures that there is a potential solution for the client to get access to the requested service.

1. check whether \mathcal{P}_A and \mathcal{C}_p entail r ,
2. if the check succeeds then **grant** access
3. else
 - (a) compute the set of disclosable credentials \mathcal{C}_D entailed by \mathcal{P}_D and \mathcal{C}_p ,
 - (b) compute a set of *missing credentials* \mathcal{C}_M out of the disclosable ones ($\mathcal{C}_M \subseteq \mathcal{C}_D$) such that:
 - \mathcal{P}_A together with \mathcal{C}_p and \mathcal{C}_M entail r , and
 - \mathcal{P}_A together with \mathcal{C}_p and \mathcal{C}_M preserve consistency.
 - (c) if a set \mathcal{C}_M exists then return \mathcal{C}_M and iterate
 - (d) else
 - i. compute a set of *excessing credentials* \mathcal{C}_E among the client's active ones ($\mathcal{C}_E \subseteq \mathcal{C}_p$) such that:
 - A. \mathcal{P}_A together with $\mathcal{C}_p \setminus \mathcal{C}_E$ preserve consistency, and
 - B. it exists $\mathcal{C}_M (\subseteq \mathcal{C}_D)$ such that:
 - \mathcal{P}_A together with $\mathcal{C}_p \setminus \mathcal{C}_E$ and \mathcal{C}_M entail r , and
 - \mathcal{P}_A together with $\mathcal{C}_p \setminus \mathcal{C}_E$ and \mathcal{C}_M preserve consistency.
 - ii. if a set \mathcal{C}_E exists then ask the client to revoke \mathcal{C}_E and iterate
 - iii. else **deny** access.

Figure 1.6: Interactive Access Control for Stateful Systems

1.4 The Contribution of this Dissertation

This work presents a framework for reasoning about interactive access control for autonomic communication. The framework describes an access control algorithm for stateless services which computes on the fly missing credentials needed for a client to get access. Further, the framework enhances the basic access control procedure to cope with stateful systems and especially with the property of non-monotonicity.

Stateful systems are essentially non-monotone and, as so, they may easily become inconsistent because of too many credentials (privileges) sent by a client. In such cases, the extended access control algorithm computes excessing (conflicting) credentials among the client's ones which make the access policy inconsistent and requests the client to revoke them.

In contrast to intra-enterprise workflow systems [7], a provider offering services has no way to assign to a client the right set of credentials which would be consisted with his future requests (simply because the provider cannot assign or prohibit the client future tasks). So, we must have a roll-back procedure by which if a user has exceeded his privileges then he has the chance to revoke them.

On top of the stateless algorithm we devise an interactive trust negotiation protocol that communicates and negotiates the missing credentials in a piecewise manner. The negotiation continues until enough trust is established and the service is granted or the negotiation fails and the process is terminated. The protocol can be run on both client and server sides so that they understand each other and automatically interoperate until the desired solution is reached or denied.

1.5 Dissertation Outline

The remainder of the dissertation is organized as follows.

Chapter 2 reviews the major architectural approaches to distributed access control. It identifies the security requirements for a possible access control architecture for autonomic network processes and summarizes them against the security requirements.

Chapter 3 presents the logical model for interactive access control. It starts with the syntax and semantics of the model followed by the description of the basic reasoning services of deduction, abduction and consistency checking. It ends with the formalization of the running example.

Chapter 4 presents the interactive access control algorithms for stateless and stateful autonomic systems. It advocates a logical framework for non-monotonic access reasoning. The chapter ends by defining the technical guarantees that the framework provides: soundness (no grant is given to unauthorized clients), completeness (authorized clients get grant) and resistance against DoS attacks.

Chapter 5 shows the correctness and completeness of the stateless and stateful algorithms. It formally defines, formulates and proves the technical guarantees of the interactive access control framework.

Chapter 6 presents the trust negotiation framework. It starts by introducing the negotiation model. Next, it shows the interactive negotiation protocol that enables two entities to automatically negotiate requirements to access a service. The chapter ends by extending the negotiation protocol with the piecewise disclosure algorithm.

Chapter 7 presents a possible architecture for interactive access control based on Business Processes for Web Services. A short introduction to Web Services is given followed by the presentation of the architecture and functional description of its components. Next, the chapter shows the implementation of the architecture using the available Web Services technologies. Then it describes the interactive access control prototype, called *iAccess*. Particularly, *iAccess* integration with the Internet security standards X.509 and SAML, and how the automated reasoning tool DLV is used as a back-end engine for the basic functionalities of deduction and abduction.

Chapter 8 concludes the dissertation and discusses potential future work.

The work presented in the dissertation has appeared in [26, 34, 31, 33, 30, 32, 27, 29, 28].

Chapter 2

Survey on Distributed Access Control

This chapter identifies the security requirements for a possible access control architecture for autonomic network processes. It reviews the major architectural approaches for distributed access control and summarizes them against the security requirements.

2.1 Authorization Requirements

The advances in communications and networking research brought distributed systems and applications to the forefront place of academic and industrial research. Authorization management has become one of most important issues concerning those systems and applications. The term *distributed authorization* attempts to comprise a whole range of issues from workflow level organizational policies, through service level management of policies, to low-level security mechanisms and architectures for enforcing authorization decisions.

We skip here the classical access control models (see e.g. [18] for a comprehensive survey) and concentrate on academic and industrial proposals for access control in distributed systems.

The basic approaches to distributed authorization, underlying all modern systems and models, are identity-based and capability-based access controls. The identity-based approach emphasizes and relies on authentication as a key property for a distributed application. It requires an entity requesting a service to be, first, securely authenticated and then the actual access control decision follows.

Taking access decisions on the basis of requester's identity becomes a liability for distributed systems offering their services in an open environment to potentially unknown clients.

Capability-based systems approach distributed authorization in a different and scalable way. Instead of relying on entity's identity they rely on user's capabilities in order to take an access decision. The term *credential* has become widely used for expressing digital access rights in a distributed environment and the *management of credentials* emerged as a key issue for a distributed authorization framework. Thus, credential-based access control [58, 50, 51, 14, 42, 43] becomes the more suitable model for enforcing distributed authorizations.

The notion of credential-based access control has also been referred in the literature as *trust management* [54], especially, in following the early papers by Blaze et al. about KeyNote, Policy-Maker and REFEREE [9, 10, 16, 11]. However, since no management of trust is actually made we prefer to use the term credential-based access control. A number of later proposals have refined the languages used for policies, single credentials or hierarchies thereof and for their evaluation [38, 39, 60, 61, 46]. However, the key focus of these proposals is usually the policy and credential language rather than the overall architecture which is responsible for the access decision. Weeks [54] offers a good survey.

The overall access control architecture is also important as we shall in the next section that no matter the trust management framework of one's choice there can be many different approaches.

Here we identify the security requirements for a possible distributed authorization architecture:

- *Separation of partner's specific security policies and requirements from the authorization system* – an autonomic process spans across many partners, each with its own security policy and requirements. Considering the high degree of autonomy of each partner, it is unrealistic to equalize (restructure) partner's security infrastructure for each inter-organizational workflow. Thus an authorization system should stay apart from the internal representation of each partner's policy and how local access decisions are taken. To do so, an authorization system should treat each partner as a distinct object encapsulating its own mechanisms for enforcement of security policy.
- *Support of different policy languages* – this requirement is closely connected with the first one and postulates that a distributed access control system should allow a service provider to define its own security policy in a language best suited for that.
- *Orchestrating requests of grant/deny/additional requirements of many different partners* – in a distributed environment where applications cross several boundaries it becomes difficult and cumbersome to impose a central authority that manages and enforces the requirements and policies of partners from different domains. Moreover some partners may not be willing to disclose their policies directly to the workflow system or even to disclose them at all. So, just combining policies from different application domains is not sufficient.
- *Client/Servent interactive communications* – a client needs a way to fulfill all partners' requirements. Privacy considerations make gathering all potentially needed credentials from a client difficult. Furthermore, this may simply be impossible. An airline may want to ask confidential information directly to its frequent fliers (e.g., confirmation of religious preferences for food) and not to the workflow system. We need a way to interactively disclose required information to the client.
- *Separate entities for policy repository and evaluation* – this simplifies the authorization server's logic and reduces the cost of access control administration.
- *Credential/Certificate-based access control* – as identified earlier in this section, it is a mechanism that requires a client to provide certificates of access rights. These certificates should be acquired before accessing a service and presented at the time of access.
- *Decentralized security administration* – each administrative domain should have full autonomy of specification, management and enforcement of its security policies.
- *Modular authorization* – allows the authorization service to work with current and future authentication and attribute services.
- *Authorization server as a separate entity* – the main objective is to decouple authorization from application logic. The authorization logic is encapsulated into an authorization service external to the application.

2.2 Access Control Architectures

The goal of this section is to survey the architectural approaches to distributed access control. It is a good starting point for a better understanding of what the basic components are in building a security architecture for autonomic business networks.

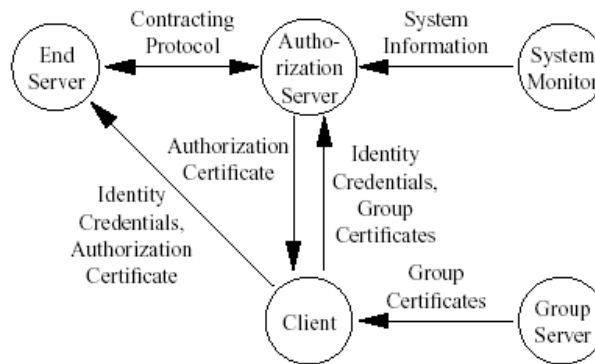


Figure 2.1: Woo and Lam Framework [55]

If we look at the proposals for distributed access control architectures [55, 24, 8, 64, 4, 53, 23, 15, 59] one of the main design features is splitting the server role into two: an *Application Server* and an *Authorization Server*, i.e. decoupling access control logic from application logic and possibly distribute the access control component [23, 59].

2.2.1 Single Policy-based Access Control

One of the earliest work on providing a general framework for expressing authorizations was proposed by Woo and Lam [56, 55]. In their work the main component of the system is an Authorization Server that performs authorization on behalf of an End Server. As shown in Figure 2.1, after a Client has requested the End Server for invoking a service, the End Server elects an Authorization Server in order to offload its access control policy for further evaluation. Then the Authorization Server takes the final access decision and hands out authorization certificates to authorized Clients. These certificates are to be forwarded by the Clients to the End Server along with their requests.

There are three more components in the framework: a System Monitor that tracks the system states; a Group Server that provides group membership information in the form of certificates (membership and nonmembership); and an Authentication Server that authenticates users during their initial sign-on, as well as, performs mutual authentication between every two entities in the system. All components and their message exchanges are shown in Figure 2.1.

The approach scales well in a distributed environment where each application server can choose its authorization server for getting an authorization decision. The idea of offloading access policies to an authorization server does not fit into the nature of autonomic processes because catering the needs of many different partners from one authorization server makes it complex and heavy in evaluating different authorization policies written in different languages.

Akenti [51] and PERMIS [15] are systems based entirely on digitally-signed documents (certificates). Figure 2.2 and Figure 2.3 show Akenti and PERMIS system components, respectively.

Akenti's flow model works as follows. When a client requests an operation he presents an X.509 identity certificate for authentication. The resource server authenticates the client and then asks the Akenti policy engine for an access decision. Akenti checks with the cache server for possibly cached certificates and if that fails, searches certificate directories across the Internet. Once Akenti has all the necessary certificates, it checks whether the client satisfies the requirements to access the resource and returns the access control decision to the resource server. The resource server then enforces the decision and returns the result back to the client. The policy engine either returns "access denied" or a list of actions that are allowed. The resource server must know how to interpret and perform the named actions.

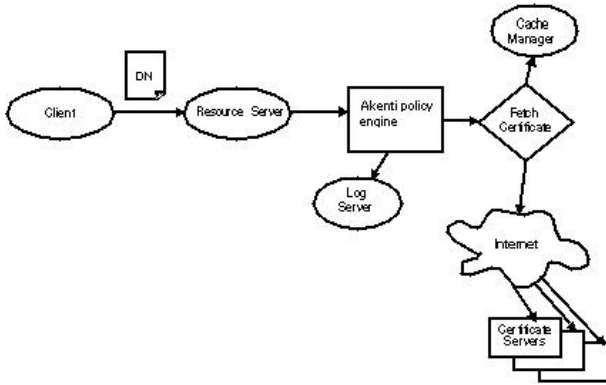


Figure 2.2: Akenti Architecture [51]

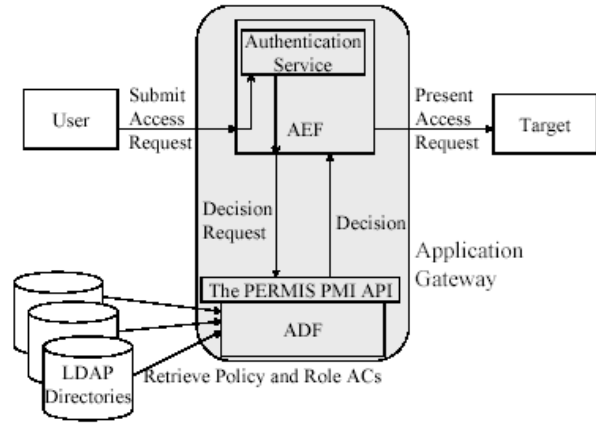


Figure 2.3: PERMIS Architecture [15]

Akenti uses three types of certificates: X.509 user identity certificates for authenticating users, use-condition certificates for specifying the conditions that must be met by a user to get access to a resource, and attribute certificates, stored on trusted servers, attesting that a user possesses specific attributes.

Each resource provider (called stakeholder) makes assertions about the conditions that must be satisfied by the user to get access to a resource. These conditions are stated in certificates signed by the stakeholders and located on a web server (local or remote) accessible by the Akenti policy engine. User attributes are asserted and signed by trusted authorities and provided as attribute certificates. So, to face the distributive nature of applications with each resource is stored a minimal authority file which contains a list of servers that supply the attribute and use-conditions certificates.

The main disadvantage of the approach is that it supports specific (ad-hoc) structure certificates for expressing policies and attributes and the centralized nature of gathering all needed certificates for getting an access decision.

The PERMIS infrastructure consists of two main subsystems: the privilege allocation and the privilege verification subsystem. The former is responsible for assigning privileges to users – issuing X.509 role assignment attribute certificates (ACs), as well as, signing and issuing a service provider's access policy as an X.509 AC. The privilege allocation subsystem stores its ACs in a (local) LDAP directory for subsequent use by the privilege verification subsystem.

The privilege verification subsystem (Figure 2.3) authenticates and authorizes a remote client as well as provides an access decision to target services. In its essence, it is divided in two subfunctions: the Access control Enforcement Function (AEF) and the Access control Decision Function (ADF). The AEF is application dependent. It authenticates the user and enforces the final access decision returned by ADF. The ADF, on the other side, is application independent so it authorizes an already authenticated user in an independent and consistent way.

Another architecture close to PERMIS is the Secure Mediator by Altenschmidt et al. [1]. Again the problem of enforcing authorization is based on digital credentials attesting user's eligibility. The Secure Mediator serves as an integrated view of variety of heterogeneous sources and access to these sources is provided via wrappers. Wrappers can be accomplished by using some distributed object managers (e.g., CORBA, software agents etc). The authors advocate a mediation protocol for secure query answering allowing clients to query resources in a direct manner (with the owner of the resource) or in an indirect way (via the mediator).

The general idea of Akenti, PERMIS and Secure Mediator projects is that the information needed for an access decision, such as identity, authorization, and attributes is stored and conveyed

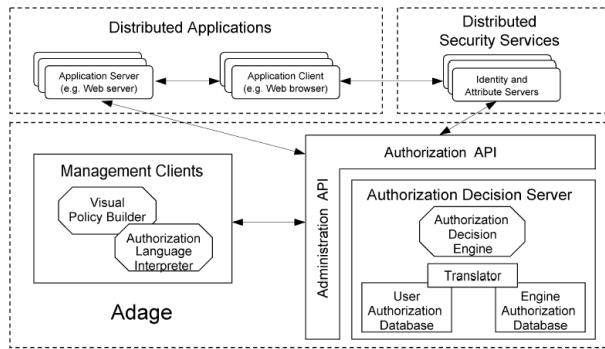


Figure 2.4: Adage System [64]

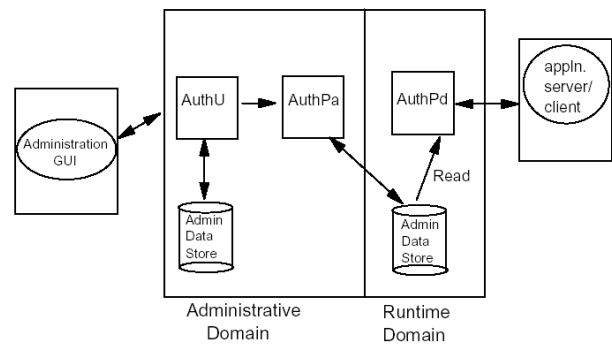


Figure 2.5: Praesidium Auth. Server [52]

in certificates, which are widely dispersed over the Internet (e.g., LDAP directories, Web servers etc.). The authorization engine has to gather and verify the certificates needed for the user's request and then evaluate them to compute an access decision.

Other two solutions that share common key principles in the design of an authorization service are Adage system [64] (Figure 2.4) and an architecture [52] (Figure 2.5) deployed by Hewlett-Packard, called Praesidium authorization server. Both approaches offer centralized security administration and modular authorization. The Application Server communicates with the Authorization Server for obtaining authorization decision. On its side, the authorization server communicates with Identity and Attribute Servers to get additional information for the client. However – here one can spot a sample feature meaningful only for architectures within one administrative domain – during the computation of the access control decision the authorization server determines whether the user needs some roles to be activated and attempts to activate them.

Both systems consist of two domains: administrative domain – concerned with setting up and management of privileges, policies, and profiles granted to principals; and runtime domain – optimized for efficient processing of authorization requests. The latter uses the information available in the administrative domain translated in a form suitable for getting fast and efficient decisions.

Because of the centralized administration of policies and privileges, each partner has to offload (reveal) its own security policies to the authorization server for translation and evaluation.

A widely discussed proposal for distributed access control is OASIS [5]. Figure 2.6 shows the interactions between a principal and an OASIS secured service. Here, before invoking a service, a principal has to obtain credentials indicating activation of specific roles. To do so, the principal contacts a role activation service, specific per domain, which issues a Role Membership Certificate (RMC) that stores all the credentials the user has activated (steps 1,2 in Figure 2.6). Role activation is carried out by the Certificate Issuing and Authentication (CIA) service on behalf of all services in a domain. After the certificate is issued, it has to be forwarded by the Client together with a request for a service (step 3) to the OASIS access control engine. In turn, the access control engine performs a procedure for certificate validation (step 3 in Figure 2.6b), by asking an appropriate CIA service, and then enforces access control on the base of client's current credentials and the authorization policy related to the requested service.

The OASIS approach allows decentralized security administration of access control policies for autonomous management domains. It encapsulates each partner's specific policy with its internal interpretation and evaluation and interoperates requirements for credentials to service level agreements.

The work by Au et al. [4] proposes a technique of using one-shot authorization tokens (Figure 2.7). A smart card is adopted as an authorization device that stores client's tokens in a secure and mobile way. In the proposed authorization scheme there are three main elements: an authorization

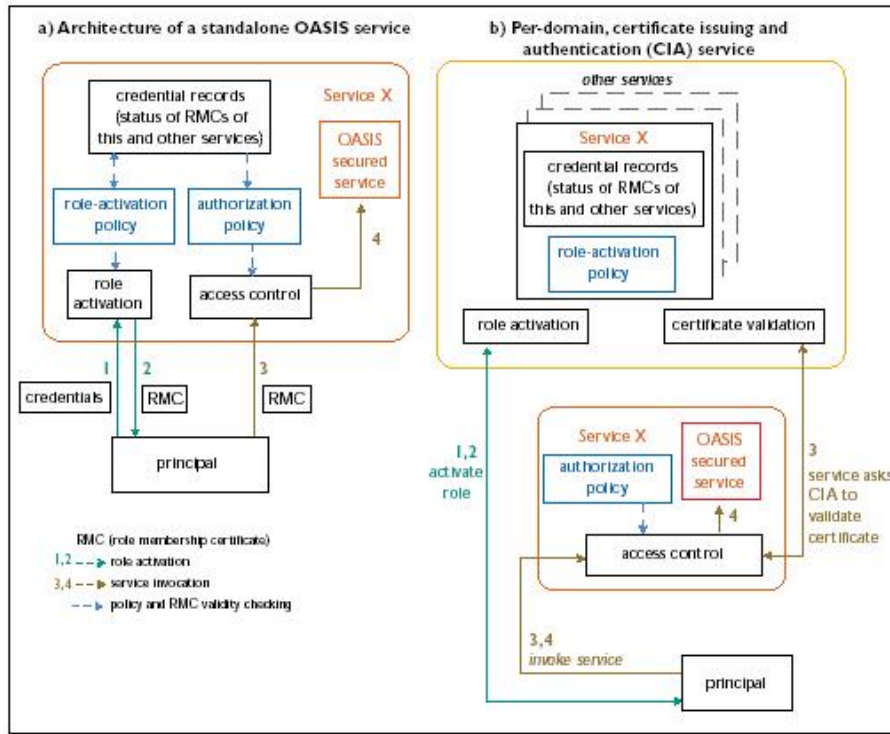


Figure 2.6: OASIS Architecture [5]

server; a client workstation; and an application server. The role of the authorization server is to provide initial credentials in the form of an authorization token to all users under the same administrative domain and to administer centrally the access control information at each application server. It is also responsible for the communications with other authorization servers from different domains in order to set up the user's initial access rights. Thus, each partner does not need to offload (reveal) its policy to the partner orchestrator of the process, but just to issue credentials in a secure token.

2.2.2 Multi-Policy Based Access Control

A step closer to orchestrating Web Services is by Beznosov et al. [8]. In this work authorizations are managed by an Authorization Service and its Access Decision Object (ADO). Figure 2.8 shows a message flow between entities in the architecture for computing an access decision.

The main advantage of the approach is the use of Policy Evaluators. Policy Evaluators serve as distinct authorities each with its own security policies. Their role is to encapsulate different authorization policies with their internal representation and evaluation – a step ahead for addressing the security requirement for separation of partner's specific security infrastructure from the authorization system.

The sequence of messages leading to an access decision is the following. An application server contacts ADO server for an authorization decision. ADO obtains references to all policy evaluators related to the client's request, asks a decision combinator for combining decisions returned by the various evaluators (according to a suitable combination policy), and returns the decision back to the application server (see Figure 2.8).

The Policy Evaluator returns to the Decision Combinator, as a result of the policy evaluation, reply *yes/no/don't know* decision. This is a step towards addressing the requirement of orches-

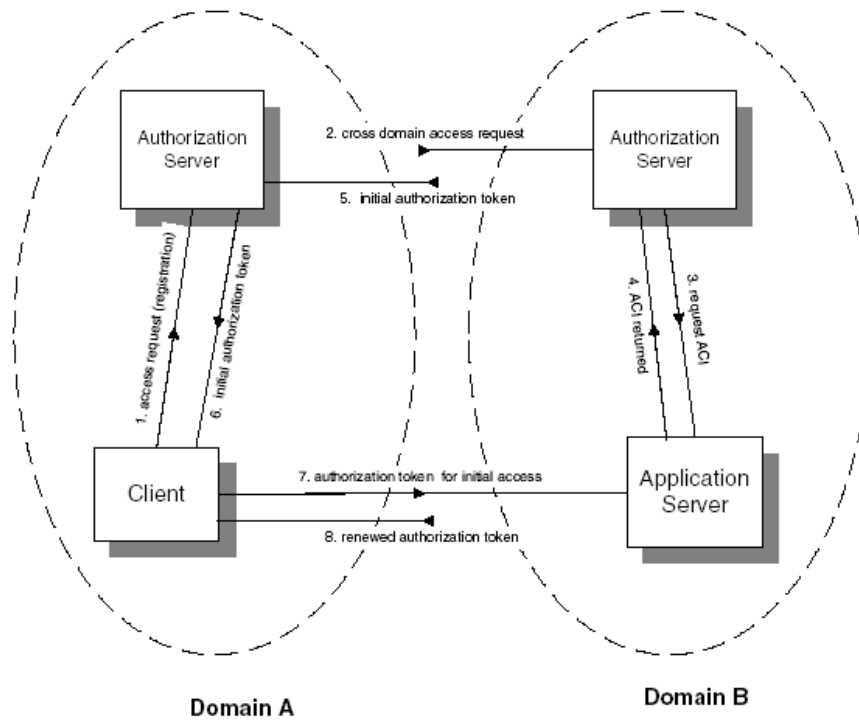


Figure 2.7: Cross-domain Authorization [4]

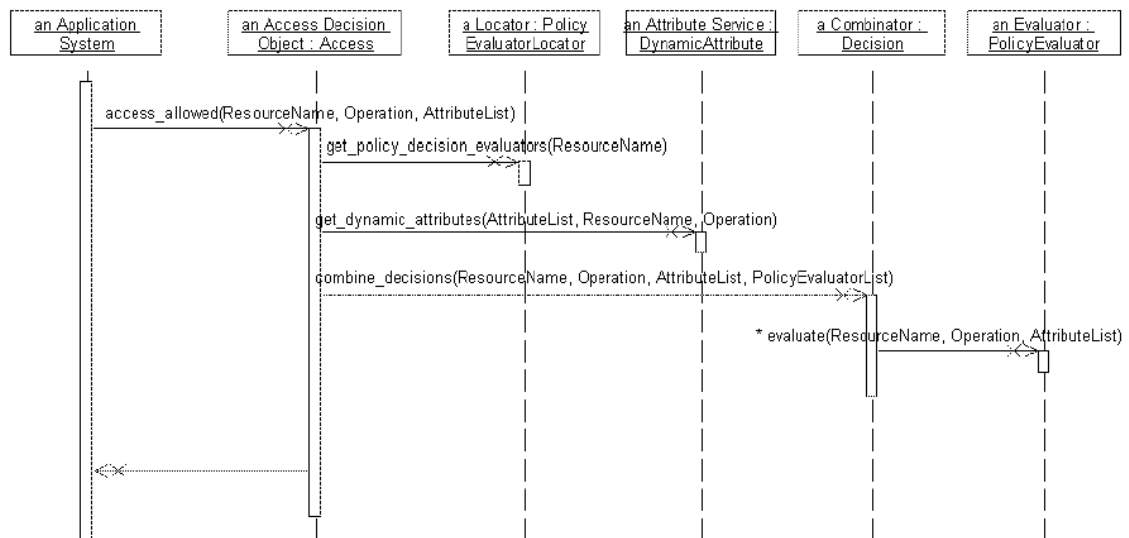


Figure 2.8: RAD Architecture [8]

trating partners' authorization requests. It will be marked in Table 2.1 as a possible approach for orchestrating authorization requests although it is not fully on target as we shall see in the next sections.

A more advanced effort for enforcing and administrating authorization policies across heterogeneous systems is the OASIS eXtensible Access Control Markup Language (XACML) framework [59]. The main actor here is the Policy Decision Point (PDP) responsible for retrieving the relevant policies wrt the client's request, evaluating them and rendering an authorization decision. The work also considers the combination of different policies from various partners using some policy

combining algorithms and getting an authorization decision on the base of evaluating them. In this case PDP has access to a partner's security policy, i.e. every partner in a process has to reveal its security policy to the PDP in order to be computed an access decision. However, we need a way to orchestrate different partners' access decisions (requests) instead of just combining their policies.

The other key approach of OASIS consortium is the Secure Assertion Markup Language (SAML) standard [44]. The main objective of the approach is to offer a standard way for exchanging authentication and authorization information between trust domains. The basic data objects of SAML are assertions. Assertions contain information that determines whether users can be authenticated or authorized to use resources. The SAML framework also defines a protocol for requesting assertions and responding to them, which makes it suitable when modeling interactive communications between entities in a distributed environment.

2.3 Access Control Systems and Security Requirements

Table 2.1 summarizes the access control systems described above and compares them against the security requirements in §2.1.

AC Systems \ Features	Auth. Server	Modular Auth.	Decentralized Security Admin.	Credential/Certificate based AC	Separate entities for policy repository and evaluation	Support different auth. languages	Combining different sec. policies	Orchest. partners' auth. requests	Client/Servent interactive comm-ns	Separation partners' spec. sec. policies from AC system
Woo&Lam	✓	✓	✓	✓	✓					
Akenti	✓		✓	✓	✓					
PERMIS	✓	✓	✓	✓	✓					
RAD	✓	✓	✓		✓	✓	✓	✓		✓
Adage	✓	✓								
Praesidium	✓	✓								
OASIS	✓	✓	✓	✓	✓	✓				✓
One-shot auth. token	✓	✓		✓	✓					✓
OASIS (XACML&SAML)	✓	✓	✓	✓	✓		✓		✓	

Table 2.1: Summary of the Access Control Systems and the Authorization Requirements

Looking at the table we find an approximation of the authorization requirements in proposals [8, 5, 59, 44].

However, there is no proper solution that captures all of the above mentioned aspects. The goal of this dissertation is to provide a model and architecture that synthesizes all those aspects into one access control framework for autonomic systems and network processes.

Chapter 3

The Logical Model

This chapter presents the logical model for interactive access control. It starts with the syntax and semantics of the model followed by description of the basic reasoning services of deduction, abduction and consistency checking, and ends with the formalization of the running example.

3.1 Syntax and Semantics

Using Datalog and logic programs for representing and reasoning about access control is customary in computer security [6, 36, 12, 7] and this work is no exception.

In the model we have the following sets of identifiers: **Role** for role identifiers; **Resource** for resource identifiers; **Action** for action identifiers performed on resources; **Attr** for attribute identifiers, where $\text{Role} \subseteq \text{Attr}$; **Issuer** for certificate issuer identifiers; and **IssuerType** for type/classification of issuers.

The table below shows the basic predicates used in the logical model.

<code>dominate (Role:r_i, Role:r_j)</code>	when Role: r_i dominates (\succeq) Role: r_j .
<code>grant (Resource:s, Action:p)</code>	when action p is granted to be performed on resource s .
<code>credential ($holder$, Attr:a, Issuer:i)</code>	a certificate attesting that $holder$ has an attribute a issued by i , where a can be a role or other property characterizing particular access rights.
<code>certificate ($subject$, Issuer:i)</code>	a certificate identifying entity $subject$ issued by i .
<code>classify (Issuer:i, IssuerType:t)</code>	classifies issuer i as a particular type t certificate authority.

Table 3.1: Predicates Used in the Logical Model

An attribute certificate is represented as a combination of two predicates: one defining the holder, attribute and the issuer of the certificate, and the other classifying the issuer authorized (trusted) to issue such attributes.

Analogously, identity certificate is represented as a predicate identifying the subject and a predicate classifying the trusted issuer.

We do not keep a set of user identifiers because in the autonomic communication's domain anybody is potentially a client and the notion of identity-based access control does not apply. Thus, *holder* and *subject* variables, shown in Table 3.1, do not have a priori fixed values but rather are used to relate with each other in order to express proper identification requirements.

Example 3 To grant access to a bank report, a client should identify itself via a trusted identity certificate and present a credential for a role bank manager (issued by the bank attribute authority).

In this situation, we are not particularly interested in the client's identity but in the relation that the subject of the identity certificate correctly maps the holder of the attribute certificate.

The model presented in this section can be adapted to *any* generic policy framework. The information we need from the underlying policy language is shown in the table above and can be found in (extracted from) most policy languages.

Another class of predicates, regarding the stateful part of the framework, are predicates describing the status of a system, shown in Table 3.2. This class of predicates, keeps track of the main activities done by users and services. In particular we have predicates specifying successful activation of services, predicates for successful completion of services; its dual one for abortion; predicates indicating granting a service to a user/role and, the opposite one, denial user's access to a service.

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

Table 3.2: Status Predicates Used in the Stateful Model

Policies are written as normal logic programs [2]. These are sets of *rules* of the form:

$$A \leftarrow B_1, \dots, B_n, \text{ not } C_1, \dots, \text{ not } C_m \quad (3.1)$$

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 $\text{ not } C_j$ is a *negative literal*, whereas the conjunction of the B_i and $\text{ not } C_j$ is called the *body* of the rule. If the body is empty the rule is called a *fact*. A normal logic program is a set of rules.

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

$$\leftarrow B_1, \dots, B_n, \text{ not } C_1, \dots, \text{ not } C_m \quad (3.2)$$

One of the most prominent semantics for normal logic programs is the *stable model semantics* proposed by Gelfond and Lifschitz [21] (see also [2] 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 (3.1) 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 (3.2) is used to rule out from the set of acceptable models situations in which B_i are true and all C_j are false (are not acceptable).

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

Definition 3.1.1 The reduct P^S of a ground logic program P with respect to a set of atoms S is the definite program obtained from P by deleting:

1. each rule that has a negative literal $\text{ not } C$ in its body with $C \in S$;
2. each negative literal in the bodies of the remaining rules.

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

Definition 3.1.2 A set of atoms S is a *stable model* of a normal logic program P iff $S = M(P^S)$.

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

1. Stable models are minimal: a proper subset of a stable model is not a stable model.
2. Stable models are grounded: each atom in a stable model has a justification in terms of the program, i.e. it is derivable from the reduct of the program with respect to the model.

Though this definition of stable models in terms of fix points is non-constructive there are constructive definitions [2] and systems [41, 35] that can cope with ground programs having tens of thousands of rules.

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

Definition 3.1.3 *The stable models of a normal logic program P with variables are those of its ground instantiation P_H with respect to its Herbrand universe¹.*

3.2 Automated Reasoning in Answer Set Programming

This section formally defines the automated reasoning services intuitively introduced in §1.3.

Definition 3.2.1 (Logical Consequence and Consistency) *Let P be a logic program and L be a ground literal. L is a logical consequence (deducible) of P , $P \models L$, if L is true in every stable model of P . P is consistent, $P \not\models \perp$, if there is a stable model for P .*

Definition 3.2.2 (Security Consequence) *A resource r is a security consequence of a policy P if (i) P is logically consistent and (ii) r is a logical consequence of P .*

Definition 3.2.3 (One-Step Deduction) *Let P be a logic program, \mathcal{A} be a set of predefined ground atoms and L be a positive literal. L is one-step deducible from \mathcal{A} according to P , $\mathcal{A} \models_1^P L$, if for some literals L_1, \dots, L_n holds:*

- (i) $L \leftarrow L_1, \dots, L_n$ is in $\mathbf{ground}(P \cup \mathcal{A})$ and
- (ii) for all credential literals L_{c_1}, \dots, L_{c_p} , $1 \leq c_i \leq n$, $\mathcal{A} \models L_{c_1} \wedge \dots \wedge L_{c_p}$.

Definition 3.2.4 (Abduction) *Let P be a logic program, H a set of ground atoms (called hypotheses or abducibles), L a ground literal (called observation), and \prec a partial order (p.o.) over subsets of H . A solution of the abduction problem $\langle L, H, P \rangle$ is a set of ground atoms E such that:*

- (i) $E \subseteq H$,
- (ii) $P \cup E \models L$,
- (iii) $P \cup E \not\models \perp$,
- (iv) any set $E' \prec E$ does not satisfy all conditions above.

Traditional partial orders are subset containment or set cardinality.

¹Essentially, we take all constants and functions appearing in the program and combine them in all possible ways. This yields the Herbrand universe. Those terms are then used to replace variables in all possible ways thus building its ground instantiation.

Definition 3.2.5 (Solution Set for a Resource r) Let P is a policy and r be a resource. A set of credentials C_S is a solution set for r according to P if r is a security consequence of P and C_S , i.e. $P \cup C_S \models r$ and $P \cup C_S \not\models \perp$.

Definition 3.2.6 (Monotonic and Non-monotonic Policy) A policy P is monotonic if whenever a set of statements C is a solution set for r according to P ($P \cup C \models r$) then any superset $C' \supset C$ is also a solution set for r according to P ($P \cup C' \models r$).

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

Definition 3.2.7 (Resource r Additive Policy) A policy P is a resource r additive if for every two sets of statements C and C' , where $C \not\subseteq C'$ and $C' \not\subseteq C$, that unlock the resource r according to P then also $C \cup C'$ unlocks r according to P .

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

Definition 3.2.8 (Resource r Subset Consistent Policy) A policy P is a resource r subset consistent if for every solution set C_S for r holds that each $C \subseteq C_S$ preserves consistency in P , i.e. $P \cup C \not\models \perp$.

Definition 3.2.9 (Well-behaved Policy) A policy P is well-behaved if for all resources $r \in P$

- (i) P is resource r additive and
- (ii) P is resource r subset consistent.

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

Proposition 3.2.1 All monotonic policies are well-behaved but the converse is not true.

Proof. In one direction the property is immediate. For the other way round we show a counter-example:

$$\begin{aligned}
 r_1 &\leftarrow C_A. \\
 r_1 &\leftarrow C_B. \\
 r_2 &\leftarrow C_C. \\
 &\leftarrow C_A, C_C. \\
 &\leftarrow C_B, C_C.
 \end{aligned}$$

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

3.3 Formalization of the Running Example

Following is the full formalization of the running example introduced in §1.2. There is a preprocessing step that validates and transforms certificates to predicates suitable for the formal model – credential (Holder: *HolderID*, Attr: *AttrName*, Issuer: *IssuerID*). Here *IssuerID* is a mapping from the trusted public keys of SOAs (Source of Authorities) to their internal policy identifiers represented by *IssuerID*. Thus, using a second predicate *classify* (*IssuerID*, *IssuerType*), it is easy

Access Policy:

- (1) `classify (planetLabClass1SOA, system).`
- (2) `classify (fraunhoferClass1SOA, institute).`
- (3) `classify (unitnClass1SOA, university).`
- (4) `assign (disk) ← authNet (*, *.unitn.it).`
- (5) `assign (disk) ← authNet (*, *.fraunhofer.de).`
- (6) `assign (disk) ← assign (run).`
- (7) `assign (run) ← authNet (193.168.205.*, *.unitn.it).`
- (8) `assign (run) ← authNet (198.162.45.*, *.fraunhofer.de).`
- (9) `assign (run) ← assign (disk), credential (*, memberPlanetLab, Issuer), classify (Issuer, system).`
- (10) `assign (run) ← assign (disk), credential (*, Attr, Issuer), classify (Issuer, university), Attr \succeq researcher.`
- (11) `assign (run) ← assign (disk), credential (*, Attr, Issuer), classify (Issuer, institute), Attr \succeq employee.`
- (12) `assign (run) ← assign (configure).`
- (13) `assign (configure) ← assign (disk), credential (*, Attr, Issuer), classify (Issuer, university), Attr \succeq assistant.`
- (14) `assign (configure) ← assign (disk), credential (*, Attr, Issuer), classify (Issuer, institute), Attr \succeq juniorResearcher.`
- (15) `assign (configure) ← authNet (*, *.it), credential (*, Attr, Issuer), classify (Issuer, university), Attr \succeq assProf.`
- (16) `assign (configure) ← authNet (*, *.de), credential (*, Attr, Issuer), classify (Issuer, institute), Attr \succeq seniorResearcher.`
- (17) `assign (configure) ← credential (*, Attr, Issuer), classify (Issuer, university), Attr \succeq fullProf.`
- (18) `assign (configure) ← credential (*, Attr, Issuer), classify (Issuer, institute), Attr \succeq boardOfDirectors.`

Disclosure Policy:

- (1) `credential (Holder, memberPlanetLab, Issuer) ← authNet (*, *.unitn.it), classify (Issuer, system).`
- (2) `credential (Holder, memberPlanetLab, Issuer) ← authNet (*, *.fraunhofer.de), classify (Issuer, system).`
- (3) `credential (Holder, employee, Issuer) ← credential (Holder, memberPlanetLab, IssuerSys),
classify (IssuerSys, system), classify (Issuer, institute).`
- (4) `credential (Holder, researcher, Issuer) ← credential (Holder, memberPlanetLab, IssuerSys),
classify (IssuerSys, system), classify (Issuer, university).`
- (5) `credential (Holder, AttrX, Issuer) ← credential (Holder, AttrY, Issuer), classify (Issuer, university),
AttrX \succ AttrY.`
- (6) `credential (Holder, AttrX, Issuer) ← credential (Holder, AttrY, Issuer), classify (Issuer, institute),
AttrX \succ AttrY.`

Figure 3.1: Planet-Lab Access and Disclosure Control Policies

to classify what SOAs are authorized (trusted) to issue particular type of attribute certificates, as shown in Figure 3.1.

Another predicate used in the example is `authNet (IP, DomainName)`. It is a tuple with first argument the IP address of the authorized network endpoint (the client's machine) and the second argument the domain name where the IP address comes from.

Following is the functional explanation of the policies shown in Figure 3.1.

The access policy:

- Rules (1), (2) and (3) classify issuers (SOAs) in different logical categories used by the access control logic. Example, Rule (1) categorizes *planetLabClass1SOA* as a system level SOA.
- Rules (4) and (5) give **disk** access to the shared network content to everybody from the University of Trento and Fraunhofer institute, regardless the IP and roles at these institutions.
- Rule (6) gives **disk** access to anybody who has a **run** access permission.
- Rules (7) and (8) allow **run** access for those machines that are internal of the two institutions (dedicated only for Planet-Lab access) and distinguished by their fixed IPs.
- Rules (9), (10) and (11) relax the previous two and allow **run** access from any place of the

institutions to those users which present either a Planet-Lab membership certificate or a role-position certificate at one of the two institutions.

- Rule (12) gives **run** access to anybody who has a **configure** access permission.
- Rules (13) and (14) give **configure** access right if a user has a **disk** access and is at minimum assistant, attested (issued) by a trusted university's SOA, or at minimum junior researcher attested by a trusted institutional SOA.
- Rules (15) and (16) relax the previous two and give **configure** access to associate professors and senior researchers provided that requests come from the respective country domains.
- Rules (17) and (18) give **configure** access regardless the geographical region only to members of board of directors and to full professors.

The disclosure policy:

- Rules (1) and (2) disclose the need for a Planet-Lab membership certificate to any request coming from domains of the respective organizations.
- Rules (3) and (4) disclose the need for an employee or a researcher certificate if either a client has already presented its Planet-Lab membership certificate or the certificate is disclosed by other rules of the disclosure policy.
- Rules (5) and (6) disclose (upgrade) the need of higher role-position certificates than those provided either by a client or (disclosed) by other rules of the policy.

Chapter 4

Interactive Access Control

This chapter presents the interactive access control algorithms for stateless and stateful autonomic systems. It advocates a logical framework for non-monotonic access reasoning. The chapter ends by defining the technical guarantees that the framework provides: soundness (no grant is given to unauthorized clients), completeness (authorized clients get grant) and resistance against DoS attacks.

4.1 Stateless Autonomic Systems

Below we summarize all the information we have recalled (policies, credentials, etc) to this extend.

\mathcal{P}_A – security policy governing access to resources,

\mathcal{P}_D – security policy controlling the disclosure of foreign (missing) credentials,

\mathcal{C}_p – set of credentials presented by a client in a single interaction,

\mathcal{C}_P – set of active credentials that have been presented by a client during an interactive access control process,

\mathcal{C}_N – set of credentials that a client has declined to present during an interactive access control process.

Now, we have all the necessary material to introduce our interactive access control algorithm. Figure 4.1 shows the protocol and algorithm for stateless services.

The intuition behind the algorithm is the following. Once the client has initiated a service request r with (optionally) a set of credentials \mathcal{C}_p , the interactive algorithm updates the client's profile of \mathcal{C}_P and \mathcal{C}_N (lines 1: and 2:). \mathcal{C}_P is updated with the newly presented credentials \mathcal{C}_p and \mathcal{C}_N is updated with the set difference of what the client was asked in the last interaction (\mathcal{C}_M) minus what he presents in the current one (\mathcal{C}_p).

Next, the algorithm consults for 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}_A according to the client's set \mathcal{C}_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}_D according to \mathcal{C}_P and from the resulting set removes all already declined and already presented credentials. The latter is used to avoid dead loops of asking something already declined or presented. Then, the algorithm computes (using abduction reasoning) all possible subsets of \mathcal{C}_D that are consistent with the access policy \mathcal{P}_A and, at the same time, grant r . Out of all these sets (if any) the algorithm selects the minimal one.

Input: r, \mathcal{C}_p
Output: grant/deny/ask(\mathcal{C}_M)

$i\text{AccessControl}(r, \mathcal{C}_p)\{$

- 1: $\mathcal{C}_P = \mathcal{C}_P \cup \mathcal{C}_p$;
- 2: $\mathcal{C}_N = \mathcal{C}_N \cup (\mathcal{C}_M \setminus \mathcal{C}_p)$, where \mathcal{C}_M is from the last interaction;
- 3: $result = i\text{AccessDecision}(r, \mathcal{P}_A, \mathcal{P}_D, \mathcal{C}_P, \mathcal{C}_N)$;
- 4: return $result$;

$\}$

$i\text{AccessDecision}(r, \mathcal{P}_A, \mathcal{P}_D, \mathcal{C}_P, \mathcal{C}_N)\{$

1. check whether r is a security consequence of \mathcal{P}_A and \mathcal{C}_P , namely
 - $\mathcal{P}_A \cup \mathcal{C}_P \models r$, and
 - $\mathcal{P}_A \cup \mathcal{C}_P \not\models \perp$.
2. if the check succeeds then return **grant** else
 - (a) compute the set of *disclosable credentials* \mathcal{C}_D as
 $\mathcal{C}_D = \{c \mid c \text{ credential that } \mathcal{P}_D \cup \mathcal{C}_P \models c\} \setminus (\mathcal{C}_N \cup \mathcal{C}_P)$,
 - (b) use abduction to find a set of *missing credentials* $\mathcal{C}_M (\subseteq \mathcal{C}_D)$ such that:
 - $\mathcal{P}_A \cup \mathcal{C}_P \cup \mathcal{C}_M \models r$, and
 - $\mathcal{P}_A \cup \mathcal{C}_P \cup \mathcal{C}_M \not\models \perp$.
 - (c) if no such set exists then return **deny** else
 - (d) return **ask**(\mathcal{C}_M).

$\}$

Figure 4.1: Interactive Access Control Algorithm for Stateless Services

Example 4 A senior researcher at Fraunhofer institute FOKUS wants to reconfigure an online service for paper submissions of a workshop. The service is part of a big management system hosted at the University of Trento’s network that is part of the Planet-Lab network, formalized in §3.3. 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 user.

Formally speaking, the request comes from a domain fokus.fraunhofer.de with an attribute credential for employee. The set of credentials is:

$$\{\text{authNet}(198.162.193.46, \text{fokus.fraunhofer.de}), \\ \text{credential}(\text{AliceMilburk}, \text{employee}, \text{fraunhoferClass1SOA})\}$$

So, according to the access policy the credentials are not enough to get **configure** access and the request would be denied (ref. rule 14 in Figure 3.1). Then, following the algorithm (step 2a in Figure 4.1) it is computed the set of disclosable credentials from the disclosure policy and the user’s set of active credentials. In our case, \mathcal{C}_P is the set of credentials mentioned above. The algorithm computes \mathcal{C}_D as the need of all roles higher in position than memberPlanetLab (ref. Figure 3.1, Disclosure Policy part). Next, abduction step (Figure 4.1 step 2b), with criterion minimal set

cardinality, computes the following missing sets that satisfy the request:

$$\begin{aligned} &\{\text{credential}(\text{AliceMilburk}, \text{juniorResearcher}, \text{fraunhoferClass1SOA})\}, \\ &\{\text{credential}(\text{AliceMilburk}, \text{seniorResearcher}, \text{fraunhoferClass1SOA})\}, \\ &\{\text{credential}(\text{AliceMilburk}, \text{boardOfDirectors}, \text{fraunhoferClass1SOA})\} \end{aligned}$$

Then, using role minimality criterion, the algorithm returns back the need for $\{\text{credential}(\text{AliceMilburk}, \text{juniorResearcher}, \text{fraunhoferClass1SOA})\}$. There is a post processing step that tracks back the internal SOA policy identifiers to their high level description that is returned back to the client.

In the next interaction, since Alice is a senior researcher, she declines to present the requested credential by returning the same query but with no entry for presented credentials ($\mathcal{C}_p = \emptyset$). So, the algorithm updates the user's profile marking the requested credential $\text{credential}(\text{AliceMilburk}, \text{juniorResearcher}, \text{fraunhoferClass1SOA})$ as declined. The difference comes when the algorithm recomputes the disclosable credentials as all disclosable credentials from the last interaction minus the newly declined one. Next, abduction computes the following sets of missing credentials that satisfy the request:

$$\begin{aligned} &\{\text{credential}(\text{AliceMilburk}, \text{seniorResearcher}, \text{fraunhoferClass1SOA})\}, \\ &\{\text{credential}(\text{AliceMilburk}, \text{boardOfDirectors}, \text{fraunhoferClass1SOA})\} \end{aligned}$$

According to minimal set cardinality criterion, the algorithm returns the need for a credential $\{\text{credential}(\text{AliceMilburk}, \text{seniorResearcher}, \text{fraunhoferClass1SOA})\}$. On the next interaction, Alice presents a certificate attesting her as a senior researcher and the algorithm grants the requested service. \square

Remark 1 Using declined credentials is essential to avoid loops in the process and to guarantee the success of interaction in presence of disjunctive information.

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

This is all we need for autonomic processes constituent by *stateless services*, in which all decisions are taken on the basis of the current set of credentials and which envisaged to be the large majority. This type of decision is characteristic of most logical approaches to access control [36, 7, 12]: we only look at the policy, the request and the set of credentials.

4.2 Stateful Autonomic Systems

Stateful systems, 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 inconsistent states, and servers should help them finding out what must be revoked and what missing to provide in order to get access to a service.

To address the problem of inconsistency, this section extends the stateless algorithm in a way that it allows a service provider to reason of not only what missing credentials are needed to get a service, but also to find out what excessing (conflicting) is among the client's set of credentials that makes the policy state inconsistent.

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

Input: $\mathcal{C}_p, \mathcal{C}_r$ and r ; **Internal input:** $\mathcal{P}_\mathcal{A}, \mathcal{P}_\mathcal{D}, \mathcal{H}, \mathcal{C}_p$;

Output: $\text{grant/deny}/\langle \text{ask}(\mathcal{C}_\mathcal{M}), \text{revoke}(\mathcal{C}_\mathcal{E}) \rangle$;

1. update $\mathcal{C}_p = (\mathcal{C}_p \setminus \mathcal{C}_r) \cup \mathcal{C}_p$,
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. check whether r is a security consequence of $\mathcal{P}_\mathcal{A}$ and \mathcal{C}_p , namely
 - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_p \models r$ and
 - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_p \not\models \perp$,
5. if the check succeeds then return **grant** else
 - (a) compute the set of *disclosable credentials* $\mathcal{C}_\mathcal{D} = \{c \mid \mathcal{P}_\mathcal{D} \cup \mathcal{C}_p \models c\} \setminus (\mathcal{C}_\mathcal{N} \cup \mathcal{C}_p)$,
 - (b) use abduction to find a minimal set of *missing credentials* $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ such that both $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_p \cup \mathcal{C}_\mathcal{M} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_p \cup \mathcal{C}_\mathcal{M} \not\models \perp$,
 - (c) if a set $\mathcal{C}_\mathcal{M}$ exists then return $\langle \text{ask}(\mathcal{C}_\mathcal{M}), \text{revoke}(\mathcal{C}_\mathcal{E}) \rangle$ and iterate else
 - i. for every $c \in \mathcal{C}_p$ introduce a new credential \hat{c} in the language,
 - ii. use abduction to find a minimal set of *missing credentials* $\mathcal{C}_\mathcal{M} \subseteq \{\hat{c} \mid c \in \mathcal{C}_p\} \cup \mathcal{C}_\mathcal{D}$ such that
 - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \{c \leftarrow \text{not } \hat{c} \mid c \in \mathcal{C}_p\} \cup \mathcal{C}_\mathcal{M} \models r$,
 - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \{c \leftarrow \text{not } \hat{c} \mid c \in \mathcal{C}_p\} \cup \mathcal{C}_\mathcal{M} \not\models \perp$,
 - 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 $\langle \text{ask}(\mathcal{C}_\mathcal{M}), \text{revoke}(\mathcal{C}_\mathcal{E}) \rangle$ and iterate.

Figure 4.2: Interactive Access Control Algorithm for Stateful Services

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}_p , the set of *active credentials* that have been presented by the client in past requests to other services within the partner's domain.

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.

Once a client initiates 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 this session context, we keep sets of *declined*, *missing* and *excessing* credentials, respectively, $\mathcal{C}_\mathcal{N}$, $\mathcal{C}_\mathcal{M}$ and $\mathcal{C}_\mathcal{E}$. 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}_p .

Our interactive access control solution for stateful services and applications is shown in Figure 4.2. 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 sets the variables $\mathcal{C}_\mathcal{N}$, $\mathcal{C}_\mathcal{M}$ and $\mathcal{C}_\mathcal{E}$.

Then, the set of active credentials \mathcal{C}_p is updated by removing the revoked ones \mathcal{C}_r from it and then adding the newly presented credentials (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}_M and \mathcal{C}_E for the next interaction output.

After the client's profile is updated, the algorithm checks whether the request r is granted by \mathcal{P}_A according to the client's set of active credentials \mathcal{C}_P (step 4). If the check fails then in step 5a the algorithm computes all credentials disclosable from \mathcal{P}_D and \mathcal{C}_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 step, it is essentially our own interactive access control algorithm for stateless services presented in §4.

If in step 7c no \mathcal{C}_M exists 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}_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}_P there is *something "wrong"* that bans the client to get any solution, i.e. it makes \mathcal{P}_A inconsistent.

In the first case there is nothing we can do and we should just 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 $\mathcal{C}_D \cup \mathcal{C}_P$ searching for a possible solution \mathcal{C}_M that unlocks r and preserves consistency in $\mathcal{P}_A \cup \mathcal{H}$. 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}_P that makes \mathcal{P}_A inconsistent. In this case we compute the set of excessing credentials \mathcal{C}_E as the set difference $\mathcal{C}_P \setminus \mathcal{C}_M$. Here we separate the definitely good (consistent) solution \mathcal{C}_M from the rest in \mathcal{C}_P .

Notice that steps 5(c)i–5(c)iii could be simplified by simply setting $\mathcal{C}_E = \mathcal{C}_P$ and $\mathcal{C}_P = \emptyset$, 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 $c \in \mathcal{C}_P$ a new symbol \hat{c} in the model, step 5(c)i. Then after obtaining the set of new symbols $\{\hat{c} \mid c \in \mathcal{C}_P\}$ we generate a set of rules $\{c \leftarrow \text{not } \hat{c} \mid c \in \mathcal{C}_P\}$. The trick here is that negating all credentials in \mathcal{C}_P , using the newly introduced symbols, and running abduction reasoning over the set union of $\{\hat{c} \mid c \in \mathcal{C}_P\} \cup \mathcal{C}_D$ (step 5(c)ii) allows us to find a *minimal* solution \mathcal{C}_M for r that itself indicates what should be revoked and what should be asked from the client.

Let us consider the set $\{c \leftarrow \text{not } \hat{c} \mid c \in \mathcal{C}_P\}$. Since we attach this set to $\mathcal{P}_A \cup \mathcal{H} \cup \mathcal{C}_P$ so it follows that all credentials in \mathcal{C}_P will be deduced except those that the corresponding new symbol appears in \mathcal{C}_M . So, all new-symbol credentials appearing in \mathcal{C}_M , computed in step 7(c)ii, will be treated such that the absence of their respective credentials in \mathcal{C}_P allows the abduction reasoning to find a solution set for r . The solution itself may contain credentials from \mathcal{C}_D and if so these credentials should be asked as missing ones from the client.

Remark 2 (Multiple Activations and Revocations of Credentials) *In an interactive access control process the algorithm may (re)ask the client to present credentials that he has revoked (was explicitly asked for that) in previous interactions or (re)ask the client to revoke credentials that the same has already activated.*

Since we do not know what solution a client has for a particular resource and in the presence of alternatives in the access policy the system may choose the "wrong" one¹ such that later on when

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

the right alternative is chosen it may require the revocation/activation of credentials that were already activated/revoked by the client in the interactions with the “wrong” one.

Example 5 *Abstracting from a specific meaning and for the sake of simplicity 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 $\mathcal{C}_P = \{C_C\}$ and history $\mathcal{H} = \emptyset$. The policies for access and disclosure control are shown below.*

Access Policy \mathcal{P}_A	Disclosure Policy \mathcal{P}_D
$r \leftarrow C_A, C_B.$	$C_A \leftarrow .$
$r \leftarrow C_C, C_D.$	$C_B \leftarrow .$
$\leftarrow C_A, C_C.$	$C_C \leftarrow .$
	$C_D \leftarrow .$

Now let suppose that the client initially requests service r with set of presented credentials $\mathcal{C}_p = \{C_A\}$.

Then according to the algorithm in Figure 4.2 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 variables and their respective values are listed in the table below.

\mathcal{H}	\mathcal{C}_P	$\mathcal{P}_A \cup \mathcal{H} \cup \{c \leftarrow \text{not } \hat{c} \mid c \in \mathcal{C}_P\}$	\mathcal{C}_D	\mathcal{C}_N	$\{\hat{c} \mid c \in \mathcal{C}_P\} \cup \mathcal{C}_D$
\emptyset	$\{C_A, C_C\}$	$r \leftarrow C_A, C_B.$ $r \leftarrow C_C, C_D.$ $\leftarrow C_A, C_C.$ $C_A \leftarrow \text{not } \hat{C}_A.$ $C_C \leftarrow \text{not } \hat{C}_C.$	$\{C_B, C_D\}$	\emptyset	$\{\hat{C}_A, \hat{C}_C, C_B, C_D\}$

The output of this step considering the minimality criterion subset containment is:

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

Here it comes the point where Remark 2 takes place. Since we do not know what credentials and especially what solution the client has in possession we must choose one of the two outcomes listed above.

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

In the other case, if we choose the solution $\langle \text{ask}(\{C_D\}), \text{revoke}(\{C_A\}) \rangle$ 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 $\mathcal{C}_P = \{C_C\}$ does not contain enough credentials to unlock r . Following that, the abduction reasoning in step 5b will not find a solution because in \mathcal{C}_P there is a credentials C_C that is inconsistent with the only solution available in $\mathcal{C}_D = \{C_A, C_B\}$.

Running again abduction reasoning with minimality criterion 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, i.e. to revoke all of his active credentials ($\mathcal{C}_E = \{C_C\} = \mathcal{C}_P$) such that he can start from an initial (consistent) state of the system.

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

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

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. check whether r is a security consequence of $\mathcal{P}_\mathcal{A}$ and $\mathcal{C}_\mathcal{P}$, namely
 - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \models r$ and
 - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \not\models \perp$,
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 $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \not\models \perp$,
 - (c) if a set $\mathcal{C}_\mathcal{M}$ exists then return $\langle \text{ask}(\mathcal{C}_\mathcal{M}), \text{revoke}(\mathcal{C}_\mathcal{E}) \rangle$ 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 min set of *miss creds* $\mathcal{C}_\mathcal{M} \subseteq \{\hat{c} \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{D}$ s.t.
 - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup (\mathcal{C}_\mathcal{P} \cap \mathcal{C}_\mathcal{U}) \cup \{c \leftarrow \text{not } \hat{c} \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{M} \models r$,
 - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup (\mathcal{C}_\mathcal{P} \cap \mathcal{C}_\mathcal{U}) \cup \{c \leftarrow \text{not } \hat{c} \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{M} \not\models \perp$,
 - 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 $\langle \text{ask}(\mathcal{C}_\mathcal{M}), \text{revoke}(\mathcal{C}_\mathcal{E}) \rangle$ and iterate.

Figure 4.3: DoS Resistant Interactive Access Control Algorithm for Stateful Services

4.3 Coping with Malicious Clients

We need to improve the stateful algorithm, shown in Figure 4.2, to protect the server against Denial of Service (DoS) attacks. To do so we consider clients as entities that can manipulate a system only via the input sets of presented \mathcal{C}_p and revoked \mathcal{C}_r credentials. Particularly we assume that a client 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 that 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 that. In both cases it would turn 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.

Example 6 (Malicious Behavior) *Considering Example 5 with the same initial conditions let us play the following scenario. Again the client submits C_A when initially requests r . Then on the next step the algorithm in Figure 4.2 returns $\langle \text{ask}(C_B), \text{revoke}(C_C) \rangle$ which is one of the two alternatives at this step and we refer to Example 5 for details. Next, since the client is a malicious one, he decides to present C_B but declines to revoke C_C in the next interaction. So, the client's set of active credentials becomes $\mathcal{C}_\mathcal{P} = \{C_A, C_B, C_C\}$ and the algorithm's output in this interaction step is $\langle \text{ask}(\emptyset), \text{revoke}(C_C) \rangle$.*

Observing the algorithm's behavior, now the client decides to revoke not only credential C_C but also his initially submitted credential C_A . After the update in the next step, the client's set of active credentials becomes $\mathcal{C}_\mathcal{P} = \{C_B\}$. According to minimality criterion subset containment the algorithm returns $\langle \text{ask}(C_A), \text{revoke}(\emptyset) \rangle$ which immediately hints the client that the set of three

credentials C_A, C_B, C_C makes the system state inconsistent. So, at this point the client can easily turn the system in a previous state by presenting C_A and C_C and the expected response by the algorithm on the next interaction is $\langle \text{ask}(\emptyset), \text{revoke}(C_C) \rangle$. \square

The improved algorithm is shown in Figure 4.3. In step 1, a new data set is introduced, the set of *revoked credentials* \mathcal{C}_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}_R the set of missing credentials \mathcal{C}_M , asked in the last interaction, and adding to the resulting set the set of newly revoked credentials \mathcal{C}_r . The motivation for the set difference $\mathcal{C}_R \setminus \mathcal{C}_M$ is that whatever the client presents from the requested set \mathcal{C}_M it is dropped from \mathcal{C}_R because it is not any more revoked and whatever it is not presented in \mathcal{C}_M but is dropped from \mathcal{C}_R is added to the set of declined credentials \mathcal{C}_N . In this case revoked and declined credentials are kept disjoint, i.e. $\mathcal{C}_R \cap \mathcal{C}_N = \emptyset$.

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

Step 2 updates the client set of active credentials by removing from \mathcal{C}_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 $\mathcal{C}_P \setminus \mathcal{C}_R$ 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 – $\mathcal{C}_p \setminus \mathcal{C}_R$ – and we add also those credentials in \mathcal{C}_p that the system has asked the client – $\mathcal{C}_p \cap \mathcal{C}_M$ – or the client has declined to present in past interactions but he presents now – $\mathcal{C}_p \cap \mathcal{C}_N$.

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 credentials² that the client has not supplied to the system.

Then, in step 4, we introduce a new data set \mathcal{C}_U . The role of \mathcal{C}_U is analogous of that for \mathcal{C}_N and serves as a data store for those credentials that a client has declined to revoke in an interaction session. \mathcal{C}_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 the 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, but he 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.

Another difference of the algorithm wrt a malicious behavior is in step 7(c)ii. Here we run the abduction reasoning over the set $\{\hat{c} \mid c \in (\mathcal{C}_P \setminus \mathcal{C}_U)\} \cup \mathcal{C}_D$. The set difference $\mathcal{C}_P \setminus \mathcal{C}_U$ 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 4.2 because whatever we drop from $\mathcal{C}_P \setminus \mathcal{C}_U$ we add it by the intersection of $\mathcal{C}_P \cap \mathcal{C}_U$.

Remark 3 Now on wherever we refer to the stateful access control algorithm we explicitly refer to its extended version shown in Figure 4.3.

²excluding the revoked credentials since the system is aware of them.

```

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

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

Figure 4.4: Global Initialization and Service Management

4.4 Stateful Session Data Management

Figure 4.4 gives the intuition of possible management of services wrt the access control decision process and its relevant data sets. The algorithm shown in Figure 4.4 works like a web server. A main 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}_p to empty sets. Of course, 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}_p so that when the client accesses again a service we just load \mathcal{C}_p .

Both \mathcal{C}_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 an empty set when a particular business process is started³. 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}_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 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 expires⁴. Even more, it is valid across multiple runnings of business processes within the entire scope of the partner's domain and its eventual deactivation depends on the partner's authorization logic. The set of active credentials \mathcal{C}_p , as shown in Figure 4.4, is updated as a side-effect of the execution of `InteractiveAccessControl` function.

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 4.4) 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 ($\subseteq \mathcal{C}_p$) 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. [20], which is the mapping of a user u onto a set of active sessions. The user session of active rights corresponds to

³It 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 $\mathcal{H} = \emptyset$

⁴The set \mathcal{C}_p is strictly time-dependent and must be periodically cleared up from already expired credentials.

$avail_session_perms(user_sessions(u:USERS))$ introduced in [20]. 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 the one in [20] because in \mathcal{C}_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. This is done by the function **Update** shown in Figure 4.4. The table below summarizes the possible updates of \mathcal{H} .

Algorithm output	Status Predicates
<i>grant</i>	grant (User: U , Service: S , Number: N), running (User: U , Service: S , Number: N)
<i>deny</i>	deny (User: U , Service: S , Number: N)
Service Execution	Status Predicates
<i>accomplished</i>	success (User: U , Service: S , Number: N)
<i>failed</i>	abort (User: U , Service: S , Number: N)

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:

$$\leftarrow \text{assign}(\text{User: } U, \text{Service: } reviewSellBids), \\ \{N.\text{success}(\text{User: } U, \text{Service: } reviewBuyBids, \text{Number: } N)\} \leq 3.$$

4.5 Technical Guarantees

This section shows the summary of the technical results that the access control algorithms provide. We refer the reader to Chapter 5 for full details on the theoretical framework.

First we define two different types of clients:

Powerful client is a client that whenever receives a request for missing and/or excessing credentials returns and/or revokes all of them.

Cooperative client is a client that whenever receives a request for missing and/or excessing credentials returns and/or revokes those of them that he has in possession.

Defining the notion of good clients wrt the stateless and stateful algorithms is still not enough to state the practical relevance of the interactive access control model. We need to introduce the notion of fairness regarding the access and disclosure control policies. We define the following two properties:

Fair Access property guarantees that whenever there is a request for a service it exists a solution in the access control policy which unlocks (grants) the service. In other words, for each resource protected by the access policy there should exist a set of credentials (a solution) that grants the resource according to the policy.

Fair Interaction property guarantees that if a solution for a service request exists (according to the access policy) then this solution should be disclosable by the disclosure control policy. In

other words, any solution for a service should be potentially disclosable to a client requesting the service.

Fair access property avoids cases where the policy specifies a solution for a service but the solution itself makes the policy state inconsistent. So even a client with the right set of credentials for the service cannot get it.

In an autonomic scenario, where a service is potentially accessible by any client, fair interaction property would disclose a solution for a service to potentially any client requesting it. So, on one side we want to be fair and disclose solutions to clients, but on the other side we want to protect/restrict the disclosure of information only to selected clients and not to anybody. To approach this problem we introduce the notion of hidden credentials.

Informally speaking, a credential is hidden if an access control system needs it for taking an access decision, but does not disclose the need to anybody. Thus, an autonomic server can dynamically protect the privacy of his policies by specifying which credentials are hidden and which are not. This allows a server to restrict access to certain services only to selected clients.

Now we can define a client with hidden credentials. A client with hidden credentials for a service is any client that has in possession the hidden credentials for that service and *knows* that these are to be pushed to the server.

The technical guarantees that the interactive framework provides are:

Termination: the stateless and stateful algorithms always terminate, i.e. in a finite number of interactions either grant or deny is returned by the algorithms (resistant against DoS attacks).

Correctness: if a client gets grant for a service then he has a solution for the service, i.e. the algorithms do not grant access to unauthorized clients.

Completeness: if a client has a solution for a service then the algorithms will grant him access.

So far, we have introduced all we need to formulate the main guarantees showing the practical relevance of the access control framework.

Completeness for a powerful client. If access and disclosure control policies guarantee fair access and interaction, respectively, then a powerful client requesting access to a service will get grant with the stateless and stateful algorithms.

Completeness for a cooperative client. If access and disclosure control policies guarantee fair access and interaction, respectively, then if a cooperative client has a solution for a service then he will get grant for the service request with the stateless and stateful algorithms.

We have the same claims for powerful and cooperative clients with hidden credentials.

Chapter 5 formally defines, formulates and proves all of the above mentioned notions and claims for the stateless and stateful algorithms, respectively.

Chapter 5

Theoretical Results

This chapter shows the correctness and completeness of the stateless and stateful algorithms. It formally defines, formulates and proves the technical guarantees of the interactive access control framework.

5.1 Stateless Framework

Definition 5.1.1 (Fair Access) *Let \mathcal{P}_A be an access control policy and let $\mathcal{C}_{\mathcal{P}_A}$ be the set of ground instances of all credentials occurring in \mathcal{P}_A . The policy \mathcal{P}_A guarantees fair access if for any service r in \mathcal{P}_A there exists a set $\mathcal{C}_S \subseteq \mathcal{C}_{\mathcal{P}_A}$ that is a solution for r .*

Definition 5.1.2 (Fair Interaction) *Let \mathcal{P}_A and \mathcal{P}_D be, respectively, an access and disclosure control policies. The policies guarantee fair interaction if*

1. \mathcal{P}_A guarantees fair access and
2. if \mathcal{C}_S is a solution for a request r then \mathcal{C}_S is disclosable by \mathcal{P}_D , i.e. $\forall c \in \mathcal{C}_S, \mathcal{P}_D \models c$.

The intuition of fair interaction is that any solution for a request should be potentially visible to clients.

Definition 5.1.3 (Powerful Client) *A powerful client is a client that whenever receives $\mathbf{ask}(\mathcal{C}_M)$ returns \mathcal{C}_M .*

Definition 5.1.4 (Cooperative Client) *A client with a set of credentials (ability) \mathcal{C} is a cooperative client if whenever receives $\mathbf{ask}(\mathcal{C}_M)$ returns $\mathcal{C}_M \cap \mathcal{C}$.*

Remark 4 *Hereinafter all \mathcal{P}_A will be well-behaved policies and all \mathcal{P}_D will be monotonic policies unless explicitly specified otherwise.*

We assume that a client initiates a service request with an empty set of presented credentials. The assumption is important to avoid initial inconsistency and to assure that a client has a successful first step.

Theorem 5.1.1 (Soundness) *Let \mathcal{P}_A be an access policy, \mathcal{P}_D be a disclosure policy and r a request. If a client gets grant r with the stateless access control algorithm then he has a solution set \mathcal{C}_S that unlocks r according to \mathcal{P}_A .*

Proof. This proof is rather straightforward. The only way to introduce a credential in $\mathcal{C}_{\mathcal{P}}$ is in line (1:) of the algorithm. Since initially $\mathcal{C}_{\mathcal{P}} = \emptyset$ so the client has sent a sequence of sets of credentials $\mathcal{C}_{p_1}, \dots, \mathcal{C}_{p_n}$ such that $\bigcup_{i=1}^n \mathcal{C}_{p_i} = \mathcal{C}_{\mathcal{P}}$. So, the client has a set of credentials that unlocks r . \square

Theorem 5.1.2 (Termination) *Let $\mathcal{P}_{\mathcal{A}}$ be an access policy, $\mathcal{P}_{\mathcal{D}}$ be a disclosure policy and r a request. The stateless access control algorithm always terminates.*

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

5.2 Completeness for Powerful and Cooperative Clients

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

We shall prove this result in stages: first for powerful and then for cooperative clients. We notice that it is fairly difficult to prove any results for non-cooperative clients: if they are unwilling to send the credentials to the server, how can ever the server grant them access?

Theorem 5.2.1 (Completeness for a Powerful Client) *Let $\mathcal{P}_{\mathcal{A}}$ be a non-monotonic access policy, $\mathcal{P}_{\mathcal{D}}$ be a disclosure policy and r a request. If $\mathcal{P}_{\mathcal{A}}$ and $\mathcal{P}_{\mathcal{D}}$ guarantee fair access and interaction then a powerful client always gets grant r with the stateless access control algorithm.*

Proof. A powerful client requests r with an initial set of presented credentials equal to empty set. The server runs steps 1 and 2 of the *iAccessControl* protocol updating client's profile of active and declined credentials. Next, *iAccessControl* runs the *iAccessDecision* function. If step 1 succeeds (no credentials are needed for r) then the algorithm returns grant at step 2 and we are done.

If step 1 does not succeed then the algorithm goes to step 2a. At this point $\mathcal{C}_{\mathcal{N}} = \mathcal{C}_{\mathcal{P}} = \emptyset$. In this case $\mathcal{C}_{\mathcal{D}}$ consists of all credentials disclosable by $\mathcal{P}_{\mathcal{D}}$. Then in step 2b the abduction algorithm will return a set $\mathcal{C}_{\mathcal{M}}$ that unlocks r because

- (i) $\mathcal{P}_{\mathcal{A}}$ guarantees fair access and so a solution set $\mathcal{C}_{\mathcal{S}}$ for r exists,
- (ii) since $\mathcal{P}_{\mathcal{D}}$ satisfies the property fair interaction so $\mathcal{C}_{\mathcal{S}}$ is disclosed by $\mathcal{P}_{\mathcal{D}}$,
- (iii) clearly $\mathcal{C}_{\mathcal{S}}$ is a subset of $\mathcal{C}_{\mathcal{D}}$, because $\mathcal{C}_{\mathcal{D}}$ consists of all credentials disclosable by $\mathcal{P}_{\mathcal{D}}$.

So, step 2c is not reached and in step 2d the algorithm returns $\text{ask}(\mathcal{C}_{\mathcal{M}})$ which satisfies the two conditions of step 2b.

If there is only one solution that unlocks r then $\mathcal{C}_{\mathcal{M}} = \mathcal{C}_{\mathcal{S}}$.

Since the client is a powerful one then on the next interaction he returns $\mathcal{C}_{\mathcal{M}}$. Then *iAccessControl* updates $\mathcal{C}_{\mathcal{P}} = \mathcal{C}_{\mathcal{M}}$ and $\mathcal{C}_{\mathcal{N}} = \emptyset$ and since $\mathcal{C}_{\mathcal{M}}$ satisfies the two conditions in step 2b from the last interaction, so it also satisfies the conditions in step 1 in the current interaction and in step 2 it returns *grant*. \square

Theorem 5.2.2 (Completeness for a Cooperative Client) *Let \mathcal{P}_A be an access policy, \mathcal{P}_D be a disclosure policy and r a request. If \mathcal{P}_A and \mathcal{P}_D guarantee fair access and interaction then if a cooperative client has a set of credentials \mathcal{C}_S that is a solution for r according to \mathcal{P}_A then the client always gets grant r with the stateless access control algorithm.*

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

Step 1.

Proof by induction on the interaction steps:

Inter. 1: Client requests service r together with an initial set of credentials $\mathcal{C}_p = \emptyset$ and we fall back exactly in the proof of Theorem 5.2.1. So, the client does not receive denial by the first interaction.

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

Inter. N+1: Now, let's suppose that the client fails to get grant r in step 1. The only way to fail is that there is no solution set in \mathcal{C}_p (i.e. there are not enough access rights). It is because in \mathcal{C}_p there are only credentials partially compiled from solutions for r , i.e. $\mathcal{C}_p \subset (\mathcal{C}_{M_1} \cup \dots \cup \mathcal{C}_{M_n})$. Also since \mathcal{P}_A is *well-behaved* so \mathcal{C}_p preserves consistency in \mathcal{P}_A . Thus the failure in step 1 could be only because of no solution for r has been presented.

Next, because \mathcal{P}_A and \mathcal{P}_D guarantee fair access and interaction so $\mathcal{C}_S \subseteq (\mathcal{C}_D \cup \mathcal{C}_p)$ but $\mathcal{C}_S \not\subseteq \mathcal{C}_p$. Then at least the set difference $\mathcal{C}_S \setminus \mathcal{C}_p$ will be computed by the abduction engine because: (i) $(\mathcal{C}_S \setminus \mathcal{C}_p) \subseteq \mathcal{C}_D$ and (ii) $(\mathcal{C}_S \cup \mathcal{C}_p) \subset (\mathcal{C}_{M_1} \cup \dots \cup \mathcal{C}_{M_n} \cup \mathcal{C}_S)$ preserves consistency (ref. Def. 3.2.9 and Def. 3.2.8). Therefore, step 2c is skipped and the client gets $\text{ask}(\mathcal{C}_M)$.

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

There is a finite number of solutions for each request r simply because \mathcal{P}_A consists of a finite number of statements. The abduction reasoning service at each interaction computes different solution wrt the solutions computed in previous interactions (simply because from the disclosable credentials we remove all presented and declined credentials from previous interactions, refer step 2a in Fig. 4.1).

Since there are finite solution sets for r and the client has one of them therefore according to what we proved in **Step 1** the client in a finite number of interactions will disclose \mathcal{C}_S , i.e. $\mathcal{C}_S \subseteq \mathcal{C}_p$, and get grant r . \square

Definition 5.2.1 (Disclosable and Hidden Credentials) *Let \mathcal{P}_D be a disclosure policy, credential c is disclosable if there is a set of credentials \mathcal{C} such that $c \notin \mathcal{C}$ and \mathcal{C} together with the disclosure policy \mathcal{P}_D entails c , namely $\mathcal{P}_D \cup \mathcal{C} \models c$. A credential c is hidden if it is not disclosable.*

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

a client must provide them when initially requests a service. Essentially, the client can work in *pull* mode with disclosable credentials and must work in *push* mode with hidden credentials.

Definition 5.2.2 (Hidden Credentials for a Resource r) *Let \mathcal{P}_A be an access control policy and \mathcal{P}_D be a disclosure control policy. A set of hidden credentials for a resource r is the set \mathcal{C}_H such that:*

1. *there exists a solution set \mathcal{C}_S for r according to \mathcal{P}_A such that $\mathcal{C}_S \supseteq \mathcal{C}_H$ and*
2. *all hidden credentials in \mathcal{C}_S wrt \mathcal{P}_D are in \mathcal{C}_H .*

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

Definition 5.2.3 (Client with Hidden Credentials for a Resource r) *A client with hidden credentials for a resource r is any client that has the set of hidden credentials \mathcal{C}_H of a solution set for r and whenever requests r he sends \mathcal{C}_H initially.*

Definition 5.2.4 (Fair Interaction with Hidden Credentials) *Let \mathcal{P}_A and \mathcal{P}_D be, respectively, an access and disclosure control policies. The policies guarantee fair interaction if*

1. *\mathcal{P}_A guarantees fair access and*
2. *if \mathcal{C}_S is a solution for a request r and \mathcal{C}_H is the set of hidden credentials for \mathcal{C}_S then the visible part of \mathcal{C}_S is disclosable by $\mathcal{P}_D \cup \mathcal{C}_H$, i.e. $\forall c \in (\mathcal{C}_S \setminus \mathcal{C}_H), \mathcal{P}_D \cup \mathcal{C}_H \models c$.*

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

The intuition behind *unfair interaction* policies is that if we have all credentials necessary to access and, even, if we know that some credentials (the hidden ones) must be sent in push mode, yet this will not be enough to get an answer from the server. We will have to push other credentials that are not needed for the access process but just to disclose the information on other missing credentials.

In the following theorems whenever we refer to the fair interaction property we implicitly link it to its version with hidden credentials.

Theorem 5.2.3 (Completeness for a Powerful Client with Hidden Credentials) *Let \mathcal{P}_A be an access policy, \mathcal{P}_D be a disclosure policy and r a request. If \mathcal{P}_A and \mathcal{P}_D guarantee fair access and interaction then a powerful client with hidden credentials \mathcal{C}_H for r always gets grant r with the stateless access control algorithm.*

Proof. A powerful client requests r with an initial set of presented credentials equal to the set of hidden credentials, i.e. $\mathcal{C}_p = \mathcal{C}_H$. Then the algorithm runs steps 1 and 2 of *iAccessControl* protocol. At this point $\mathcal{C}_p = \mathcal{C}_H$ and $\mathcal{C}_N = \emptyset$.

Next, if step 1 succeeds, \mathcal{C}_H itself is a solution set for r , then the algorithm returns grant at step 2 and we are done.

If step 1 does not succeed then the algorithm goes to step 2a. In this case \mathcal{C}_D consists of all credentials disclosable by \mathcal{P}_D and \mathcal{C}_H because $\mathcal{C}_P = \mathcal{C}_H$. Then in step 2b the abduction algorithm will return a set \mathcal{C}_M because at least one such set exists:

- (i) \mathcal{P}_A and \mathcal{P}_D guarantee fair access and interaction and so for the solution set \mathcal{C}_S , corresponding to the set of hidden credentials \mathcal{C}_H that the client has, its visible part $\mathcal{C}_S \setminus \mathcal{C}_H$ will be disclosed by \mathcal{P}_D and \mathcal{C}_H (ref. Def. 5.2.4),
- (ii) clearly $(\mathcal{C}_S \setminus \mathcal{C}_H) \subseteq \mathcal{C}_D$, because \mathcal{C}_D consists of all credentials disclosable by $\mathcal{P}_D \cup \mathcal{C}_H$,
- (iii) according to Definition 5.2.2 the set $\mathcal{C}_H \subseteq \mathcal{C}_S$ and so the set $\mathcal{C}_S \setminus \mathcal{C}_H$ together with \mathcal{C}_P preserve consistency in \mathcal{P}_A (simply because $(\mathcal{C}_S \setminus \mathcal{C}_H) \cup \mathcal{C}_P = \mathcal{C}_S$ is a solution set),

So, step 2c is not reached and in step 2d the algorithm returns $\text{ask}(\mathcal{C}_M)$ which satisfies the two conditions of step 2b.

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

Since the client is a powerful client then on the next interaction he returns \mathcal{C}_M . Then the algorithm updates $\mathcal{C}_P = \mathcal{C}_H \cup \mathcal{C}_M$ and $\mathcal{C}_N = \emptyset$ and because \mathcal{C}_M satisfies the two conditions in step 2b, from the last interaction, so it also satisfies the conditions in step 1 and in step 2 it returns *grant*. With this we finish the proof. \square

Theorem 5.2.4 (Completeness for a Cooperative Client with Hidden Credentials) *Let \mathcal{P}_A be an access policy, \mathcal{P}_D be a disclosure policy and r a request. If \mathcal{P}_A and \mathcal{P}_D guarantee fair access and interaction then if a cooperative client with hidden credentials \mathcal{C}_H for r has a solution set \mathcal{C}_S for r wrt the set \mathcal{C}_H (i.e., $\mathcal{C}_S \supseteq \mathcal{C}_H$) then the client always gets grant r with the stateless access control algorithm.*

Proof. Analogously of theorem 5.2.2 we will prove it in two steps. First step, by induction, showing that in a single interaction if the client does not get grant r then he gets $\text{ask}(\mathcal{C}_M)$. In other words, he will never receive denial by the algorithm. Second step, we will show that if the first step is true then the client with a solution set \mathcal{C}_S always gets grant r .

Proof by induction on the interaction steps:

Inter. 1: Client requests service r together with an initial set of credentials $\mathcal{C}_P = \mathcal{C}_H$ and we fall back exactly in the proof of Theorem 5.2.3.

Inter. N: Here we use the induction hypothesis that the client fails to get grant r and gets $\text{ask}(\mathcal{C}_M)$ in the previous interaction. Now, suppose that the client fails to get grant r in step 1. Then the only difference with the respective interaction step in the proof of Theorem 5.2.2 is that the client's solution set \mathcal{C}_S is still disclosable by \mathcal{P}_D and \mathcal{C}_P because \mathcal{C}_H is already in the set \mathcal{C}_P , see Definition 5.2.2. So, according to fair interaction property follows that $\mathcal{C}_S \setminus \mathcal{C}_H$ is visible and the abduction service at least finds it as a solution.

The rest of the proof can be done along the same line as in Theorem 5.2.2. \square

5.3 Stateful Framework

This section shows the correctness and completeness of the interactive access control algorithm for stateful systems (shown in Figure 4.3).

Now on we assume that the sets of missing and excessing credentials are disjoint, i.e., $\mathcal{C}_M \cap \mathcal{C}_E = \emptyset$. If it does not hold then the client will reject to answer the server's request. We note that this requirement holds by construction of the interactive algorithm (ref. Figure 4.3). We also assume that at any time in an interaction process the sets of currently presented and revoked credentials are also disjoint, i.e., $\mathcal{C}_p \cap \mathcal{C}_r = \emptyset$.

In the following we extend the powerful and cooperative clients with the notion of compliancy.

Definition 5.3.1 (Powerful and Compliant Client) *A powerful and compliant client is a client that whenever receives $\langle \text{ask}(\mathcal{C}_M), \text{revoke}(\mathcal{C}_E) \rangle$ returns $\langle \mathcal{C}_M, \mathcal{C}_E \rangle$, i.e. activates all $c \in \mathcal{C}_M$ and revokes any $c \in \mathcal{C}_E$.*

Definition 5.3.2 (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 $\langle \text{ask}(\mathcal{C}_M), \text{revoke}(\mathcal{C}_E) \rangle$ returns $\langle \mathcal{C}_M \cap \mathcal{C}, \mathcal{C}_E \cap \mathcal{C} \rangle$, i.e. activates all $c \in (\mathcal{C}_M \cap \mathcal{C})$ and revokes any $c \in (\mathcal{C}_E \cap \mathcal{C})$.*

Proposition 5.3.1 *If a client has a set of credentials \mathcal{C} then in all executions of the algorithm his set of active credentials \mathcal{C}_P never exceeds \mathcal{C} , i.e. $\mathcal{C}_P \subseteq \mathcal{C}$.*

Proposition 5.3.2 *If a client has a set of credentials \mathcal{C} then in all executions the excessing credentials \mathcal{C}_E , he could be potentially asked, are among those in \mathcal{C} , i.e. $\mathcal{C}_E \subseteq \mathcal{C}$.*

Proof. The proof is trivial and it follows straightforwardly from the algorithm's structure. Step 7(c)iiiA computes excessing credentials from the set of active credentials \mathcal{C}_P , i.e. $\mathcal{C}_E \subseteq \mathcal{C}_P$. Then according to Proposition 5.3.1 follows that $\mathcal{C}_E \subseteq \mathcal{C}$. \square

Proposition 5.3.3 *For any client with the set of credentials \mathcal{C} is valid that $\mathcal{C}_E \cap \mathcal{C} = \mathcal{C}_E$.*

Lemma 5.3.1 *Let \mathcal{P}_A be an access policy, \mathcal{P}_D be a disclosure policy and r a request. If the stateful access control algorithm returns $\langle \text{ask}(\mathcal{C}_M), \text{revoke}(\mathcal{C}_E) \rangle$ then either $\mathcal{C}_M \neq \emptyset$ or $\mathcal{C}_E \neq \emptyset$.*

Proof. We prove the claim in two parts. First one for step 7c and second part for step 7(c)iiiB.

First part: if we assume that $\mathcal{C}_M = \emptyset$ in step 7c, meaning the client has enough access rights, then we immediately contradict with step 7b, which shows that the client does not have enough access rights. So, $\mathcal{C}_M \neq \emptyset$.

Second part: let assume that $\mathcal{C}_E = \emptyset$ in step 7(c)iiiB. It means that there is no new symbol credential $\hat{c} \in \mathcal{C}_M$, where \mathcal{C}_M is computed in step 7(c)ii. Therefore $\mathcal{C}_M \subseteq \mathcal{C}_D$ and thus we have that $\mathcal{P}_A \cup \mathcal{H} \cup (\mathcal{C}_P \cap \mathcal{C}_U) \cup \{c \leftarrow \text{not } \hat{c} \mid c \in (\mathcal{C}_P \setminus \mathcal{C}_U)\} \cup \mathcal{C}_M \models r$. However, since $\mathcal{C}_M \subseteq \mathcal{C}_D$ this implies that $\mathcal{P}_A \cup \mathcal{H} \cup (\mathcal{C}_P \cap \mathcal{C}_U) \cup \{c \leftarrow \text{not } \hat{c} \mid c \in (\mathcal{C}_P \cup \mathcal{C}_U)\} \cup \mathcal{C}_M \models c$ for all $c \in \mathcal{C}_P$ because \hat{c} is a new symbol not occurring in \mathcal{P}_A and \mathcal{H} . Therefore, $\mathcal{P}_A \cup \mathcal{H} \cup \mathcal{C}_P \cup \mathcal{C}_M \models r$ and $\mathcal{P}_A \cup \mathcal{H} \cup \mathcal{C}_P \cup \mathcal{C}_M \not\models \perp$ which contradicts with the assumption in step 7b that no such set \mathcal{C}_M exists. \square

Theorem 5.3.1 (Soundness) *Let \mathcal{P}_A be an access policy, \mathcal{P}_D be a disclosure policy and r a request. If a client gets grant r with the stateful access control algorithm then he has a solution set \mathcal{C}_S that unlocks r according to \mathcal{P}_A .*

Proof. Let suppose that the client, requested service r , got *grant*. Then the only way to introduce a credential in \mathcal{C}_P is by step 2 of the algorithm. Since initially $\mathcal{C}_P = \emptyset$ so the client has sent a sequence of sets of credentials $\mathcal{C}_{p_1}, \dots, \mathcal{C}_{p_n}$ such that $\bigcup_{i=1}^n \mathcal{C}_{p_i} = \mathcal{C}_P$. Then the client has a set of credentials that unlocks it. \square

Theorem 5.3.2 (Termination) *Let \mathcal{P}_A be an access policy, \mathcal{P}_D be a disclosure policy and r a request. The stateful access control algorithm always terminates.*

Proof. The algorithm terminates when either *grant* or *deny* is returned back.

We will prove the termination by well-founded tuple ordering. At each interaction we associate a tuple $\langle \mathcal{C}_1, \dots, \mathcal{C}_n \rangle$ of credentials and stipulate that $\langle \mathcal{C}_1, \dots, \mathcal{C}_n \rangle \prec \langle \mathcal{C}'_1, \dots, \mathcal{C}'_n \rangle$ if either $\mathcal{C}_1 \subset \mathcal{C}'_1$ or for some $i < n$ $\mathcal{C}_1 = \mathcal{C}'_1, \mathcal{C}_2 = \mathcal{C}'_2, \dots, \mathcal{C}_i = \mathcal{C}'_i$ and $\mathcal{C}_{i+1} \subset \mathcal{C}'_{i+1}$.

This is a well-founded ordering which has as a top element the tuple with all credentials occurring in the policies. We associate the tuple $\langle \mathcal{C}_N, \mathcal{C}_U, \mathcal{C}_P \cup \mathcal{C}_R \rangle$ to each interaction. At the end of the interaction, which does not return grant or deny, we have already proved by Lemma 5.3.1 that $\mathcal{C}_M \neq \emptyset$ or $\mathcal{C}_E \neq \emptyset$. At the next interaction we have also received \mathcal{C}_p and \mathcal{C}_r . We use $\langle \mathcal{C}_N, \mathcal{C}_U, \mathcal{C}_P \cup \mathcal{C}_R \rangle$ to denote the previous interaction and the primed version for the values at the end of the current interaction.

If $\mathcal{C}_M \setminus \mathcal{C}_p \neq \emptyset$ then $\mathcal{C}_N = \mathcal{C}_N \cup (\mathcal{C}_M \setminus \mathcal{C}_p)$ is strictly increasing and thus $\langle \mathcal{C}_N, \mathcal{C}_U, \mathcal{C}_P \cup \mathcal{C}_R \rangle \prec \langle \mathcal{C}_N \cup (\mathcal{C}_M \setminus \mathcal{C}_p), \mathcal{C}_U', \mathcal{C}_P' \cup \mathcal{C}_R' \rangle$ and we are done.

if $\mathcal{C}_M \setminus \mathcal{C}_p = \emptyset$ and $\mathcal{C}_E \setminus \mathcal{C}_r \neq \emptyset$ then $\mathcal{C}_N = \mathcal{C}_N \cup (\mathcal{C}_M \setminus \mathcal{C}_p)$ is unchanged but $\mathcal{C}_U = \mathcal{C}_U \cup (\mathcal{C}_E \setminus \mathcal{C}_r)$ is strictly increasing and thus $\langle \mathcal{C}_N, \mathcal{C}_U, \mathcal{C}_P \cup \mathcal{C}_R \rangle \prec \langle \mathcal{C}_N \cup (\mathcal{C}_M \setminus \mathcal{C}_p), \mathcal{C}_U \cup (\mathcal{C}_E \setminus \mathcal{C}_r), \mathcal{C}_P' \cup \mathcal{C}_R' \rangle$ and we are done.

Let us now consider the case $\mathcal{C}_M \setminus \mathcal{C}_p = \emptyset$ and $\mathcal{C}_E \setminus \mathcal{C}_r = \emptyset$. If $\mathcal{C}_r = \mathcal{C}_E$ and $\mathcal{C}_M = \mathcal{C}_p$ then by construction the algorithm returns grant. So, we must be in a situation where $\mathcal{C}_r \supset \mathcal{C}_E$ or $\mathcal{C}_p \supset \mathcal{C}_M$, i.e. we have either given something we have not been asked or revoked something we have not been asked to revoke.

Since \mathcal{C}_N and \mathcal{C}_U remain unchanged therefore we have to prove that $\mathcal{C}_P \cup \mathcal{C}_R \subset \mathcal{C}_P' \cup \mathcal{C}_R'$ where according to Figure 4.3 we have

$$\begin{aligned} (1) \quad \mathcal{C}_R' &= (\mathcal{C}_R \setminus \mathcal{C}_M) \cup (\mathcal{C}_r \cap \mathcal{C}_E) \\ (2) \quad \mathcal{C}_P' &= (\mathcal{C}_P \setminus \mathcal{C}_R') \cup (\mathcal{C}_p \setminus \mathcal{C}_R') \cup (\mathcal{C}_p \cap \mathcal{C}_M) \cup (\mathcal{C}_p \cap \mathcal{C}_N) \end{aligned}$$

To prove it first we will show that $\forall c \in \mathcal{C}_P \cup \mathcal{C}_R : c \in \mathcal{C}_P' \cup \mathcal{C}_R'$ and second that $\exists c \in \mathcal{C}_P' \cup \mathcal{C}_R' : c \notin \mathcal{C}_P \cup \mathcal{C}_R$.

To prove that $\forall c \in \mathcal{C}_P \cup \mathcal{C}_R : c \in \mathcal{C}_P' \cup \mathcal{C}_R'$ let us assume the converse $\exists c \in \mathcal{C}_P \cup \mathcal{C}_R : c \notin \mathcal{C}_P' \cup \mathcal{C}_R'$ and we will show that it contradicts with some facts. Let $c \in \mathcal{C}_P$ then since $c \notin \mathcal{C}_P'$ follows that according to step (2) $c \in \mathcal{C}_R'$ but then $c \in \mathcal{C}_P' \cup \mathcal{C}_R'$ which is contradiction.

Let $c \notin \mathcal{C}_P$ and $c \in \mathcal{C}_R$. Since $c \notin \mathcal{C}_R'$ and by step (1) follows that $c \notin \mathcal{C}_R \setminus \mathcal{C}_M$ so $c \in \mathcal{C}_M$. Since $\mathcal{C}_p \supset \mathcal{C}_M$ so $c \in \mathcal{C}_p \cap \mathcal{C}_M$ and then from step (2) follows that $c \in \mathcal{C}_P'$. Thus $c \in \mathcal{C}_P' \cup \mathcal{C}_R'$ which is again contradiction.

We have just proved that $\forall c \in \mathcal{C}_P \cup \mathcal{C}_R : c \in \mathcal{C}_P' \cup \mathcal{C}_R'$.

Let us now resume the conditions we have. We are in a case $\mathcal{C}_p \supset \mathcal{C}_M$ or $\mathcal{C}_r \supset \mathcal{C}_E$ where either $\mathcal{C}_p \supseteq \mathcal{C}_M$ or $\mathcal{C}_r \supseteq \mathcal{C}_E$.

To prove the second part that $\exists c \in \mathcal{C}_P' \cup \mathcal{C}_R' : c \notin \mathcal{C}_P \cup \mathcal{C}_R$ we will divide it in the following cases and prove each of them respectively:

1. if $\mathcal{C}_r = \mathcal{C}_E$ and $\mathcal{C}_p \setminus \mathcal{C}_M \subseteq \mathcal{C}_R$ then we know that $\forall c \in \mathcal{C}_p \setminus \mathcal{C}_M$ and step (2) follows that $c \notin \mathcal{C}_P'$ and thus the algorithm returns grant.

This case describes the situation where a client presents as additional credentials some of those already revoked by him and so the system will not consider them when taking an access decision (they do not change \mathcal{C}_P).

2. if $\mathcal{C}_r = \mathcal{C}_E$ and $(\mathcal{C}_p \setminus \mathcal{C}_M) \setminus \mathcal{C}_R \subseteq \mathcal{C}_P$ then we still get grant as $\forall c \in (\mathcal{C}_p \setminus \mathcal{C}_M) \setminus \mathcal{C}_R$ holds that $c \notin \mathcal{C}_R'$ (ref. step (1)) and following that we get $c \in \mathcal{C}_P'$ (ref. step (2)), but it was still previously consistent with the policy and so the algorithm returns grant.

Here we extend case 1 and reason for those credentials, additionally presented by the client, that have already been activated by him. In this case these credentials do not influence on the access decision since they do not change $\mathcal{C}_{\mathcal{P}}$.

3. if $\mathcal{C}_r = \mathcal{C}_{\mathcal{E}}$ and $\mathcal{C}_p \setminus \mathcal{C}_{\mathcal{M}} \not\subseteq \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{R}}$ then $\exists c \in \mathcal{C}_p \setminus \mathcal{C}_{\mathcal{M}} : c \notin \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{R}}$ then according to step (1) and assumption that $\mathcal{C}_p \cap \mathcal{C}_r = \emptyset$ follows that $c \notin \mathcal{C}_{\mathcal{R}}'$ and so in step (2) this credential is added to $\mathcal{C}_{\mathcal{P}}'$. Therefore it holds that $\mathcal{C}_{\mathcal{P}}' \cup \mathcal{C}_{\mathcal{R}}' \supset \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{R}}$.
4. if $\mathcal{C}_r \supset \mathcal{C}_{\mathcal{E}}$ then $\forall c \in \mathcal{C}_r \setminus \mathcal{C}_{\mathcal{E}}$ follows that
 - if $c \notin \mathcal{C}_{\mathcal{R}} \setminus \mathcal{C}_{\mathcal{M}}$ then $c \notin \mathcal{C}_{\mathcal{R}}'$ and so it does not change $\mathcal{C}_{\mathcal{P}}'$ and thus the proof follows from the previous cases 1, 2 and 3 with $\mathcal{C}_r = \mathcal{C}_{\mathcal{E}}$.
 - if $c \in \mathcal{C}_{\mathcal{R}} \setminus \mathcal{C}_{\mathcal{M}}$ then it will also not change $\mathcal{C}_{\mathcal{R}}'$ and $\mathcal{C}_{\mathcal{P}}'$ from the previous interaction and we can go to the previous cases 1, 2 and 3 with $\mathcal{C}_r = \mathcal{C}_{\mathcal{E}}$.

So, we proved that in an interaction process a client either increases the set of declined credentials $\mathcal{C}_{\mathcal{N}}$ or, if it remains unchanged, he increases the set of not revoked credentials $\mathcal{C}_{\mathcal{U}}$ or, if both of them remain unchanged, the client increases his sets of total credentials¹ submitted to the system. The process continues until the client exhausts all credentials occurring in the access policy.

With this we finish the proof. \square

5.4 Completeness for Powerful and Cooperative Compliant Clients

Theorem 5.4.1 (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 stateful access control algorithm.*

Proof. Essentially along the lines of the stateless access control algorithm (shown in Figure 4.1). \square

Of course the whole business of 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 monotonic policy 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, we inherit and extend the assumption from the stateless framework so that, now, whenever a client initially requests a service he submits $\mathcal{C}_p = \mathcal{C}_r = \emptyset$ and if hidden credentials $\mathcal{C}_{\mathcal{H}}$ are needed then $\mathcal{C}_p = \mathcal{C}_{\mathcal{H}}$ and $\mathcal{C}_r = \emptyset$.

Here one could doubt why we do not relieve the assumption of arbitrary sets of initially presented and revoked credentials. Anyway it will not influence on the proofs since whatever is in the initial sets \mathcal{C}_p and \mathcal{C}_r we can add \mathcal{C}_p to $\mathcal{C}_{\mathcal{P}}$ and drop \mathcal{C}_r from $\mathcal{C}_{\mathcal{P}}$ and restart the proof with the new set $\mathcal{C}_{\mathcal{P}}$ assuming that the client initially presents $\mathcal{C}_p = \mathcal{C}_r = \emptyset$.

Theorem 5.4.2 (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 $\mathcal{P}_{\mathcal{A}} \cup \mathcal{H}^2$ and $\mathcal{P}_{\mathcal{D}}$ guarantee fair access and interaction then a powerful and compliant client always gets grant r with the stateful access control algorithm.*

¹Credentials that are either revoked or active.

²It 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, etc.

Proof. Let us have a powerful client that requests r with an initial set of presented and revoked credentials equal to an empty set, namely $\mathcal{C}_p = \mathcal{C}_r = \emptyset$. Also we assume that the set of active credentials is a non-empty set containing credentials that the client has submitted in previous interactions with services within the same partner's domain, $\mathcal{C}_p \neq \emptyset$.

So, keeping the assumption of the initial empty sets, steps 1 to 5 compute $\mathcal{C}_p = \mathcal{C}_r = \mathcal{C}_R = \mathcal{C}_N = \mathcal{C}_U = \mathcal{C}_M = \mathcal{C}_E = \emptyset$. If the check in step 6 succeeds then in step 7 the algorithm returns grant (either no credentials are needed for r or the client already has a solution in \mathcal{C}_p for r) and we are done.

If step 6 does not succeed then the algorithm goes to step 7a. Then using the assumption of fair access and interaction follows that a solution \mathcal{C}_S for r exists, it is disclosable by \mathcal{P}_D and $(\mathcal{C}_S \setminus \mathcal{C}_p) \subseteq \mathcal{C}_D$.

Now that we are sure of existence of a solution for r , there are two cases:

- Step 7b finds a missing set \mathcal{C}_M and so $\langle \text{ask}(\mathcal{C}_M), \text{revoke}(\mathcal{C}_E) \rangle$ is sent back to the client. In this case $\mathcal{C}_E = \emptyset$.

Then on the next interaction step, since the client is a powerful one, he presents all $c \in \mathcal{C}_M$. After the updates, in step 6, since $\mathcal{C}_p = \mathcal{C}_p \cup \mathcal{C}_M$ then the check succeeds and in the next step 7 the algorithm returns *grant* r .

- Step 7b does not find a missing set \mathcal{C}_M . In this case, since at least one solution \mathcal{C}_S exists in \mathcal{C}_D , follows that in \mathcal{C}_p there are “wrong” credentials that ban the client to get a solution, namely $\mathcal{C}_p \setminus \mathcal{C}_S \neq \emptyset$. So, the algorithm goes to step 7(c)ii. At this step the abduction reasoning does compute a missing set \mathcal{C}_M simply because, at least, one such exists and is $\mathcal{C}_M = \{\hat{c} \mid c \in (\mathcal{C}_p \setminus \mathcal{C}_S)\} \cup (\mathcal{C}_S \setminus \mathcal{C}_p)$ such that $\mathcal{C}_M \subseteq \{\hat{c} \mid c \in \mathcal{C}_p\} \cup \mathcal{C}_D$ and $\mathcal{C}_E = \{c \mid \hat{c} \in \mathcal{C}_M\} \neq \emptyset$ conforming to Lemma 5.3.1. We have the set difference of $\mathcal{C}_p \setminus \mathcal{C}_S$ and $\mathcal{C}_S \setminus \mathcal{C}_p$ just to assure that we do not ask the client to revoke and, at the same time, activate any credentials from the solution \mathcal{C}_S .

Here $\mathcal{P}_A \cup \mathcal{H} \cup \{c \leftarrow \text{not } \hat{c} \mid c \in \mathcal{C}_p\} \cup \mathcal{C}_M$ is equivalent to $\mathcal{P}_A \cup \mathcal{H} \cup \mathcal{C}_S$. Since \mathcal{C}_S is a solution for r follows that abduction finds at least this missing set and step 7(c)iii is *skipped*. Then in step 7(c)iiiA the new \mathcal{C}_E and \mathcal{C}_M are computed. We remark that the just computed two sets replaced back in the equations of step 7(c)ii transform them into $\mathcal{P}_A \cup \mathcal{H} \cup (\mathcal{C}_p \setminus \mathcal{C}_E) \cup \mathcal{C}_M \models r$ and $\not\models \perp$.

Then after the result $\langle \text{ask}(\mathcal{C}_M), \text{revoke}(\mathcal{C}_E) \rangle$ is returned back to the client, since he is powerful, he activates all missing and revokes all excessing credentials. On the next interaction, in step 6, since \mathcal{C}_p is updated to $\mathcal{C}_p = (\mathcal{C}_p \setminus \mathcal{C}_E) \cup \mathcal{C}_M$ follows that the deduction check succeeds and in step 7 the algorithm returns *grant* r .

We note that the algorithm grants a powerful client in no more than two interactions. With this we finish the proof. \square

Theorem 5.4.3 (Completeness for a Cooperative and Compliant Client) *Let \mathcal{P}_A be an access policy, \mathcal{P}_D be a disclosure policy, \mathcal{H} be history of executions and r a request. If $\mathcal{P}_A \cup \mathcal{H}$ and \mathcal{P}_D guarantee fair access and interaction then if a cooperative and compliant client has a set of credentials \mathcal{C}_S that is a solution for r according to \mathcal{P}_A then the client always gets grant r with the stateful access control algorithm.*

Proof. We will prove the theorem in two parts. First part by induction, showing that in a single interaction if a cooperative client does not get *grant* r then he gets $\langle \text{ask}(\mathcal{C}_M), \text{revoke}(\mathcal{C}_E) \rangle$. In other words he will never receive denial of the algorithm. Second part, we will show that because of the first part and Theorem 5.3.2 for termination a cooperative client with a solution set \mathcal{C}_S always gets *grant* r .

Part I Proof by induction on the interaction steps:

Inter. 1: A cooperative client requests r with initial sets $\mathcal{C}_p = \mathcal{C}_r = \emptyset$. Since the client has a solution \mathcal{C}_S for r and because of fair access and interaction properties so at least one solution exists and is disclosable by \mathcal{P}_D , i.e. $(\mathcal{C}_S \setminus \mathcal{C}_p) \subseteq \mathcal{C}_D$.

Let assume that the client does not get *grant* r in step 7 of the algorithm and that abduction in step 7b cannot find a missing set \mathcal{C}_M . Then the step 7(c)iii of denial is *skipped* because at least one solution exists, which is $\mathcal{C}_M = \{\hat{c} \mid c \in (\mathcal{C}_p \setminus \mathcal{C}_S)\} \cup (\mathcal{C}_S \setminus \mathcal{C}_p)$. This solution satisfies the conditions in step 7(c)ii and can be checked in the same way as in the Theorem 5.4.2. After that steps 7(c)iiiA and 7(c)iiiB compute and return back to the client $\langle ask(\mathcal{C}_M), revoke(\mathcal{C}_E) \rangle$.

Inter. N: Here we use the abduction hypothesis that the client fails to get *grant* r and gets $\langle ask(\mathcal{C}_M), revoke(\mathcal{C}_E) \rangle$ in the previous interaction. Now, suppose that the client fails to get *grant* r in step 7 and in step 7b the abduction reasoning fails to find a missing set.

So, up to now in the client's set of \mathcal{C}_p there are “wrong” credentials that ban the client to get a missing set (solution) for r . Since the client has a solution \mathcal{C}_S for r and because of fair access and interaction properties so this solution is disclosable by \mathcal{P}_D . Here, it is also the point where we use the fact that \mathcal{P}_D is a *monotonic* policy. It is because in the set of active credentials \mathcal{C}_p we have credentials partially compiled from requests for other solutions for r which the client does not have. And so if the policy \mathcal{P}_D is non-monotonic we cannot guarantee that the solution \mathcal{C}_S is always disclosable (in the presence of other credentials).

Again a missing set $\mathcal{C}_M = \{\hat{c} \mid c \in (\mathcal{C}_p \setminus \mathcal{C}_S)\} \cup (\mathcal{C}_S \setminus \mathcal{C}_p)$ exists which analogously of Theorem 5.4.2 can be checked that it satisfies the conditions in step 7(c)ii and so denial is skipped. We only point out that since the client is cooperative then the sets \mathcal{C}_U and \mathcal{C}_N will never overlap with the credentials the client has presented to the system, i.e. $\mathcal{C}_U \cap \mathcal{C}_p = \emptyset$ and $\mathcal{C}_N \cap \mathcal{C}_p = \emptyset$. And so the result $\langle ask(\mathcal{C}_M), revoke(\mathcal{C}_E) \rangle$ in step 7(c)iiiB is returned back to the client.

Part II What we have so far:

- If the client does not get *grant* he gets $\langle ask(\mathcal{C}_M), revoke(\mathcal{C}_E) \rangle$ (just proved in Part I),
- The client is never idle, i.e. never gets $\langle ask(\emptyset), revoke(\emptyset) \rangle$ (follows from Lemma 5.3.1),
- The algorithm at certain interaction always returns *grant* or *deny* (Theorem 5.3.2),

According to the three cases above follows that the client in a finite number of interactions will get *grant* r . \square

Theorem 5.4.4 (Completeness for a Powerful and Compliant Client with Hidden Credentials) *Let \mathcal{P}_A be an access policy, \mathcal{P}_D be a disclosure policy, \mathcal{H} be history of executions and r a request. If $\mathcal{P}_A \cup \mathcal{H}$ and \mathcal{P}_D guarantee fair access and interaction then a powerful client with hidden credentials \mathcal{C}_H for r always gets *grant* r with the stateful access control algorithm.*

Proof. The proof of the theorem is analogously of that for Theorem 5.4.2. The only difference is that initially the client presents the set of hidden credentials \mathcal{C}_H that helps him to disclose the visible part of a solution \mathcal{C}_S from \mathcal{P}_D , i.e. $\forall c \in \mathcal{C}_S \setminus \mathcal{C}_H : \mathcal{P}_D \cup \mathcal{H} \models c$ (ref. Definitions 5.2.2 and 5.2.4). So, in step 7b because of fair access and interaction properties follows that $\mathcal{C}_S \subseteq (\mathcal{C}_p \cup \mathcal{C}_D)$ and the abduction reasoning potentially can find a missing set \mathcal{C}_M .

The rest of the proof follows (can be done along the same line) from Theorem 5.4.2. With this we finish the proof. \square

Theorem 5.4.5 (Completeness for a Cooperative and Compliant Client with Hidden Credentials) *Let \mathcal{P}_A be an access policy, \mathcal{P}_D be a disclosure policy, \mathcal{H} be history of executions and r a request. If $\mathcal{P}_A \cup \mathcal{H}$ and \mathcal{P}_D guarantee fair access and interaction then if a cooperative and compliant client with hidden credentials \mathcal{C}_H for r has a solution set \mathcal{C}_S for r wrt the set \mathcal{C}_H (i.e., $\mathcal{C}_S \supseteq \mathcal{C}_H$) then the client always gets grant r with the stateful access control algorithm.*

Proof. Let us start from the fact that the client has a set of hidden credentials \mathcal{C}_H for r and the same has also a solution set \mathcal{C}_S wrt \mathcal{C}_H . It follows that, first, according to Definition 5.2.2 the set $\mathcal{C}_H \subseteq \mathcal{C}_S$ – important step assuring that the client has the right set of hidden credentials *especially* for the solution \mathcal{C}_S . Second, according to Definition 5.2.4 the solution set \mathcal{C}_S is disclosable by \mathcal{P}_D and \mathcal{C}_H .

Then using the assumption that whenever the client requests a service he initially presents the hidden credentials, we can construct the same proof along the one for Theorem 5.4.3. The only difference is that since the hidden credentials are initially submitted so at least the solution \mathcal{C}_S is disclosable by \mathcal{P}_D . Next, using fair access and interaction properties the algorithm returns $\langle ask(\mathcal{C}_M), revoke(\mathcal{C}_E) \rangle$ until the client discloses the entire \mathcal{C}_S .

With this we finish the proof. □

Chapter 6

Trust Negotiation

This chapter presents the trust negotiation framework. It starts by introducing the negotiation model. Next, it shows the interactive negotiation protocol that enables two entities to automatically negotiate requirements to access a service. Then, the chapter ends by extending the negotiation protocol with the piecewise disclosure algorithm.

6.1 The Trust Negotiation Model

Interactive access control allows a server to compute on the fly missing credentials needed to grant access and to adapt its responses on the basis of client's presented and declined credentials. Yet, it may disclose too much information on *what credentials a client needs*. Automated trust negotiation allows for a controlled disclosure on *what credentials a client has* during a mutual disclosure process. Yet, it requires pre-arranged policies and sophisticated strategies. How do we bootstrap from simple security policies a comprehensive interactive trust negotiation scheme that combines the best of both worlds without their limitations? This is the subject of the chapter.

In an autonomic network scenario servers must have a way to find out what credentials are required for clients to get access to resources. Clients, once asked for the missing credentials, may be unwilling to disclose them unless the server discloses some of its credentials first, i.e. negotiate the need of sensitive credentials.

Solution for the first problem is our interactive access control algorithm presented in §4. Solution for the second problem is trust negotiation, as advocated by Yu, Winslett et al. [63, 62] and Bonatti and Samarati [12]. Here, we can structure a security policy to specify what credentials a client must have already shown to get access to our own credentials, i.e. we specify the sequence of disclosable credentials that gradually establish trust.

The two problems are related but different. For sake of example consider the viewpoint of a server. In the first one, we help the server to compute the *missing set* of (foreign) credentials that a client needs to get access to a service. The second approach helps the server, in response to some counter requests, to control the disclosure of its own credentials by stipulating what (*foreign*) credentials a client must supply to get access to the server's local ones.

Both approaches in their core have limitations: the first approach does not allow for a piece-meal disclosure of what clients eventually need. The second one requires sophisticated and rigid structuring of policies to work.

If we merge the two frameworks we have the following open issues:

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 missing is 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
- (a) Bob must have a way to request in a piecewise fashion the missing credentials.

6.2 Access vs Negotiation Control

To combine automated trust negotiation and interactive access control we assume that both a client and a server have the three logical security policies:

1. a policy for access to *own* resources \mathcal{P}_{AR} on the basis of *foreign* credentials,
2. a policy for access to *own* credentials \mathcal{P}_{AC} on the basis of *foreign* credentials,
3. a policy for disclosure the need of (missing) *foreign* credentials \mathcal{P}_D .

Technically speaking we could merge 1 and 2 into a flat policy for protecting sensitive resources as in [62, 63]. 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. The motivation mainly comes from the fact that requirements for negotiation of sensitive credentials are influenced by security and privacy needs and as so their 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 but needed for the sake of privacy.

Forcing separation between policies 1 and 2 we gain another advantage wrt the existing approaches: we free the access policy from the restrictions usually posed on the policy for access to credentials and thus addressing the domain of non-monotonic reasoning. The access control policy can 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 [37]). We only need deduction and its sister abduction.

The policy for access to credentials we keep monotonic because of its particular nature: a client and server in a negotiation process mutually exchange agreements on credentials needed to establish trust in each other. Usually, those credentials and sequences of exchanging them have arbitrary nature since the client and the sever have autonomic view on their own policies and requirements. Thus \mathcal{P}_{AC} can easily become inconsistent if non-monotonic properties are posed on it and, as so, assuming non-monotonicity becomes counter-intuitive from the negotiation purpose point of view.

6.3 The Negotiation Protocol

As already mentioned, we explicitly distinguish between an access control process and a negotiation control process. The distinction reflects the level of management of credentials. We keep a separate set of active credentials for the access control process for request r , denoted by \mathcal{C}_r , and the overall set of active credentials for the negotiation process, denoted by \mathcal{C}_P . We also keep a set of declined

credentials, denoted by \mathcal{C}_N , and a set of counter-requested credentials that are still in a negotiation process (not yet terminated), denoted by \mathcal{C}_{neg} . The last set of credentials is used to avoid dead loops of recursively running the negotiation protocol again and again on the same counter requests.

Figure 6.1 shows our interactive trust negotiation protocol. The *iAccessControl* function works like a web server. The server initiates a new *access control session* (runs *iAccessControl*) when a client initially submits a request for a service. Then, each counter-request from the client side is checked whether it is already being negotiated by the server (step 3). If yes, then *iAccessControl* associates the request to the potential output of the already running thread (step 8). If the request is not in a negotiation then *iAccessControl* updates \mathcal{C}_{neg} with r and runs the *TrustNegotiationProtocol* in a new thread (steps 4 and 5). After a successful negotiation *iAccessControl* removes r from \mathcal{C}_{neg} .

The intuition behind the negotiation protocol is the following:

- A client, Alice, sends a service request r and (optionally) a set of credentials \mathcal{C}_p to a server, Bob (the *TrustNegotiationProtocol* input).
- Once the protocol is initiated, it updates the over all active credentials with the newly presented ones, step 1, and if the request r is for a service then it updates \mathcal{C}_r as well.
- 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}_{AR} , his policy for disclosure of foreign credentials, \mathcal{P}_D , the set of credentials \mathcal{C}_r and the set of declined credentials \mathcal{C}_N (step 5).
- If r is a request for a credential then he calls *iAccessDecision* with his policy for access to own credentials, \mathcal{P}_{AC} , again his policy for disclosure of foreign credentials, \mathcal{P}_D , the set of overall active credentials \mathcal{C}_P and the set of declined credentials \mathcal{C}_N (step 7).
- In the case of computed missing credentials \mathcal{C}_M (step 8), Bob checks whether \mathcal{C}_M was computed from the service request (step 9). If yes then, in step 10, Bob adds to \mathcal{C}_r those credentials from \mathcal{C}_M that have already been negotiated with Alice. Next, Bob removes the negotiated credentials from \mathcal{C}_M to avoid duplicate negotiations for already committed credentials (step 11). Steps 10 and 11 assure that if some of the missing credentials (computed for r) have been negotiated (agreed) by Bob and Alice so they should be taken into account without duplicate negotiations.
- In step 13, Bob transforms \mathcal{C}_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}_M . Alice will run the same protocol swapping roles.
- When Bob's process receives all responses, it checks whether the current process is for the service request r and if it holds then Bob updates \mathcal{C}_r with those missing credentials that have been successfully negotiated and presented by Alice (step 14). Step assuring separation of credentials between access and negotiation control.
- Next, Bob checks whether the missing credentials have been supplied by Alice (step 15).
- If \mathcal{C}_M was not reached, Bob restarts the loop and consults the *iAccessDecision* algorithm for a new decision.
- When a final decision of grant or deny is taken, a respective response (steps 19 or 21) is sent back to Alice.

Technicality in the protocol is in the way the server requests missing credentials back to the client. 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 under the main access control process. At that point of the protocol, it is important that each of the finished threads updates presented and declined set of

```

Global vars:  $\mathcal{C}_N, \mathcal{C}_P, \mathcal{C}_r, \mathcal{C}_{neg}$ . Initially,  $\mathcal{C}_N = \mathcal{C}_P = \mathcal{C}_r = \mathcal{C}_{neg} = \emptyset$ ;
iAccessControl{
  1:  $r = \text{receiveRequest}()$ ;
  2:  $\mathcal{C}_p = \text{receiveCredentials}()$ ;
  3: if  $r \notin \mathcal{C}_{neg}$  then
  4:    $\mathcal{C}_{neg} = \mathcal{C}_{neg} \cup \{r\}$ ;
  5:   run TrustNegotiationProtocol( $r, \mathcal{C}_p$ ) in a new thread;
  6:    $\mathcal{C}_{neg} = \mathcal{C}_{neg} \setminus \{r\}$ ;
  7: else
  8:   associate the request for  $r$  to the output of the respective running thread;
}
TrustNegotiationProtocol( $r, \mathcal{C}_p$ ){
  1:  $\mathcal{C}_P = \mathcal{C}_P \cup \mathcal{C}_p$ ;
  2: if isService( $r$ ) then  $\mathcal{C}_r = \mathcal{C}_r \cup \mathcal{C}_p$ ;
  3: repeat
  4:   if isService( $r$ ) then
  5:      $result = iAccessDecision(r, \mathcal{P}_{AR}, \mathcal{P}_D, \mathcal{C}_r, \mathcal{C}_N)$ ;
  6:   else // isCredential( $r$ )
  7:      $result = iAccessDecision(r, \mathcal{P}_{AC}, \mathcal{P}_D, \mathcal{C}_P, \mathcal{C}_N)$ ;
  8:   if  $result == \text{ask}(\mathcal{C}_M)$  then
  9:     if isService( $r$ ) then
  10:       $\mathcal{C}_r = \mathcal{C}_r \cup (\mathcal{C}_M \cap \mathcal{C}_P)$ ;
  11:       $\mathcal{C}_M = \mathcal{C}_M \setminus \mathcal{C}_P$ ;
  12:    fi
  13:    AskCredentials( $\mathcal{C}_M$ );
  14:    if isService( $r$ ) then  $\mathcal{C}_r = \mathcal{C}_r \cup (\mathcal{C}_M \cap \mathcal{C}_P)$ ;
  15:    if  $\mathcal{C}_M \subseteq \mathcal{C}_P$  then  $result = \text{grant}$ ;
  16:  fi
  17: until  $result == \text{grant}$  or  $result == \text{deny}$ ;
  18: if  $result == \text{grant}$  and isCredential( $r$ ) then
  19:   sendResponse( $r$ );
  20: else
  21:   sendResponse( $result$ );
}
AskCredentials( $\mathcal{C}_M$ ){
  1: parfor each  $c \in \mathcal{C}_M$  do
  2:   sendRequest( $c$ );
  3:   if receiveResponse() ==  $c$  then  $\mathcal{C}_P = \mathcal{C}_P \cup \{c\}$  else  $\mathcal{C}_N = \mathcal{C}_N \cup \{c\}$ ;
  4: done
}

```

Figure 6.1: The Trust Negotiation Protocol

credentials appropriately. So, we avoid the situations where some running parallel threads ask the client already asked credentials or already declined ones computed in other running threads under the same main process.

Also an important point here is to clarify the way we treat declined and not yet released credentials. In a negotiation process, declining a credential is when an entity is asked for it and the same entity replies to the same request with answer deny. In the second case, when the entity is asked for a credential and, instead of reply, there is a counter request for more credentials, then the

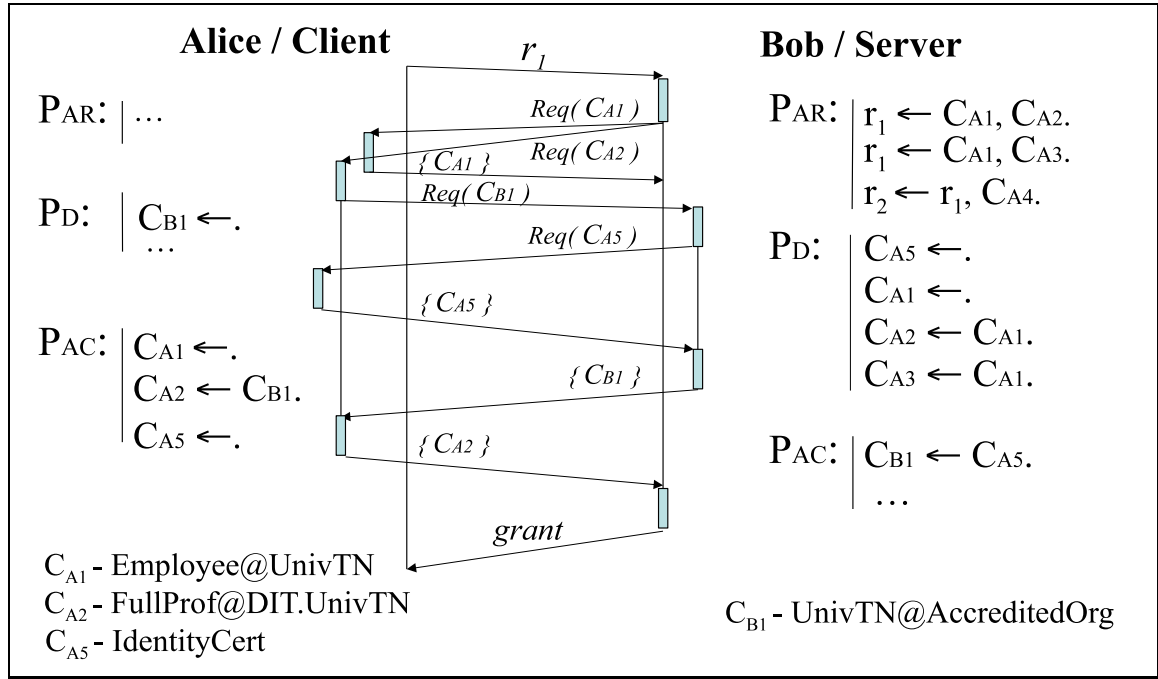


Figure 6.2: Example of Interoperability of the Negotiation Protocol

thread, started the original request, awaits the client for an explicit reply and treats the requested credential as not yet released. In any case, at the end of a the negotiation process the client either supplies the originally asked credential or declines it.

The thread based implementation (with shared \mathcal{CP} and \mathcal{CN}) is necessary to allow for a *polynomial execution time* of the trust negotiation protocol in the number of queries to the abduction algorithm. Indeed, without a shared memory for received credentials it is possible to structure the policies in a way that a credential will be asked far too many times. In this way the protocol queries to \mathcal{P}_{AC} are bounded by the number of credentials occurring in the policy.

Example 7 Figure 6.2 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. [63] where the Alice's local credentials are marked with subscript "A" and Bob's with "B", respectively. Bob's access policy \mathcal{P}_{AR} says that access to resource r_1 is granted if $\{C_{A1}, C_{A2}\}$ or, alternatively, $\{C_{A1}, C_{A3}\}$ are presented by Alice. To get access to r_2 Alice should satisfy the requirements for access to r_1 and present C_{A4} .

Analogously, we read Bob's disclosure policy as to disclose the need for a credential C_{A2} there should be already disclosed a credential C_{A1} , which by default is always disclosable. But in contrast, the need for a credential C_{A4} is never disclosed but expected by the policy \mathcal{P}_{AR} when r_2 is requested. It is an example of a hidden credential.

The real interactions start when Alice requests r_1 from Bob. Then, suppose that the set $\{C_{A1}, C_{A2}\}$ is minimal wrt the other alternative $\{C_{A1}, C_{A3}\}$, let say that C_{A2} contains a role lower in the hierarchy than the role in C_{A3} . So, Bob replies with two counter requests to Alice.

Alice, in her turn, runs the two requests in new threads and replies to Bob, according to her policy for access to sensitive credentials \mathcal{P}_{AC} , with a counter request for C_{B1} and the disclosure of C_{A2} .

The negotiation process continues, as shown in the figure, until Bob discloses all credentials requested by Alice and Alice, in her turn, discloses all credentials requested by Bob so that at the end the desired resource is granted.

Some possible interpretations of credentials are shown in the bottom of the figure. □

However, we have not solved the problem of piecewise disclosure of missing foreign credentials yet. This turns out to be also possible as we shall see in the next section.

6.4 Piecewise Disclosure

```

TrustNegotiationProtocol( $r, C_p$ ) { ... }
AskCredentials( $C_M$ ) {
  1: while  $C_M \not\subseteq (C_P \cup C_N)$  do
  2:    $C_{M1} = \text{PiecewiseDisclosure}(C_M)$ ;
  3:   if  $C_{M1} == \perp$  then return;
  4:   parfor each  $c \in C_{M1}$  do
  5:     sendRequest( $c$ );
  6:     if receiveResponse() ==  $c$  then  $C_P = C_P \cup \{c\}$  else  $C_N = C_N \cup \{c\}$ ;
  7:   done
  8: done
}
PiecewiseDisclosure( $C_M$ ) {
  1:  $C_{D1} = \{c \mid C_P \models_1^{\mathcal{P}_D} c\} \setminus (C_P \cup C_N)$ ;
  2:  $\mathcal{P}_{D1} = \{\hat{c} \leftarrow B \mid c \leftarrow B \in \mathcal{P}_D\} \cup \{c \leftarrow \hat{c} \mid c \leftarrow B \in \mathcal{P}_D \text{ and } c \notin (C_{D1} \cup C_N)\}$ ;
  3:  $Q = \{q \leftarrow \bigwedge_{c \in C_M} c\}$ ;
  4:  $result = \text{abduction}(q, C_{D1}, \mathcal{P}_{D1} \cup C_P \cup Q)$ ;
  5: return  $result$ ;
}

```

Figure 6.3: The Piecewise Negotiation Protocol

The intuition here is that Bob may not want to disclose the missing credentials all at once or directly to Alice. Instead he may want to ask Alice first some less sensitive credentials¹ assuring him that Alice is enough trustworthy to disclose her other credentials and so on until the missing ones are disclosed.

To address this issue we extend the protocol in Section 6.3 with an algorithm for piecewise 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 one step deduction (Definition 3.2.3) over the disclosure policy \mathcal{P}_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 piecewise disclosure algorithm, see Figure 6.3. With its new version the **AskCredentials** function (Figure 6.3) takes as input the set of missing credentials C_M (as the old one) and internally loads the policy for disclosure control \mathcal{P}_D that C_M was computed from. In a nutshell, the algorithm requests the client all missing credentials supplied in the input, but with the difference of stepwise awaiting for each of the computed steps by the **PiecewiseDisclosure** algorithm. In other words, when a next step of missing credentials is computed (step 2) the algorithm waits 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 C_{M1} comprises two cases:

¹Here we point out that the stepwise approach may require the client to provide credentials that are not directly related to a specific resource but needed for a fine-grained disclosure control.

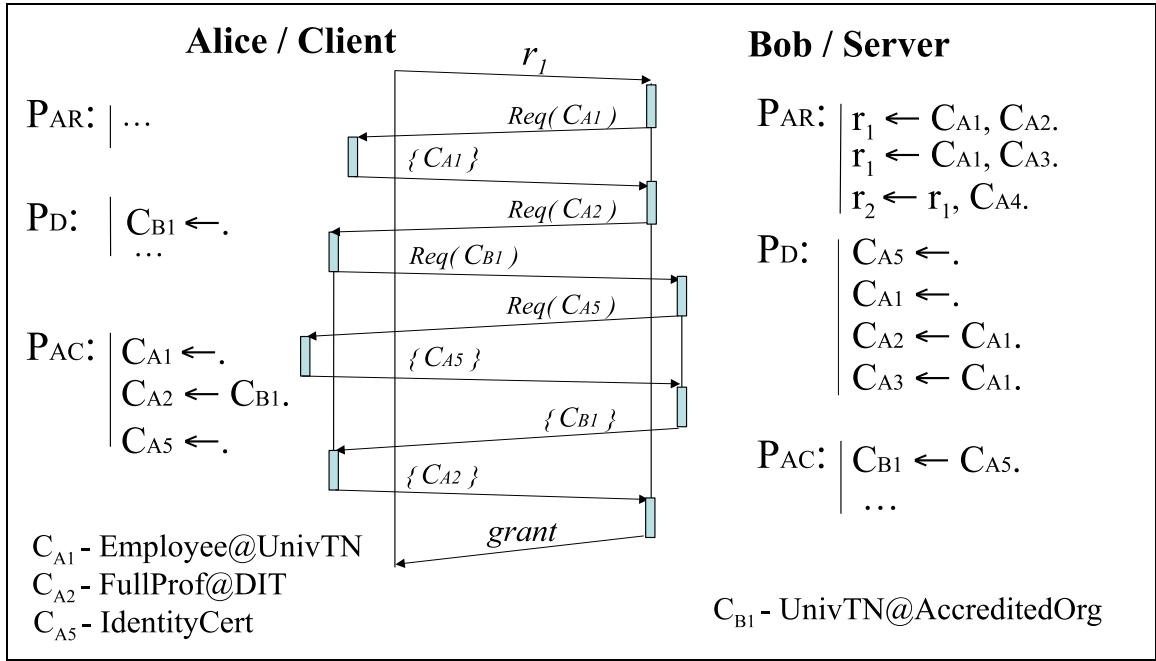


Figure 6.4: Example of Piecewise Negotiation of Credentials

either the set of presented credentials $\mathcal{C}_{\mathcal{D}}$ 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 **PiecewiseDisclosure** is to determine at each interaction step exactly the relevant credentials that are needed to reach at the end the set $\mathcal{C}_{\mathcal{M}}$. For doing so, we compute the set of disclosable credentials by one step deduction over $\mathcal{P}_{\mathcal{D}}$, set $\mathcal{C}_{\mathcal{D}1}$ in step 1. 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) $\mathcal{C}_{\mathcal{M}}$ credentials are derived (step 3). Additionally, we also change syntactically the structure of $\mathcal{P}_{\mathcal{D}}$ rule so that relevant credentials in $\mathcal{C}_{\mathcal{D}1}$ must be abduced and can no longer be derived from chaining other rules (step 2).

We do that by changing a rule of the from $c \leftarrow c_1, \dots, c_n$ into a pair of rules $\hat{c} \leftarrow c_1, \dots, c_n$ 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 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 c as a primitive atom (if it is actually needed to derive q , i.e. either some of the $c \in \mathcal{C}_{\mathcal{M}}$ or others needed for a fine-grained disclosure control).

Example 8 Figure 6.4 shows an example of how the piecewise disclosure algorithm, displayed in Figure 6.3, works. Here we have the same scenario as in Example 7. Now, Bob, instead of returning back to Alice the need of $\{C_{A1}, C_{A2}\}$, he requests the credentials in a piecewise fashion. Instead of asking for an employee certificate and full professor certificate, Bob first asks for the employee one (C_{A1}), the less sensitive one, and then on successful negotiation for it, he discloses the need for a full professor certificate (C_{A2}). In this case Bob assures himself that the information for the more sensitive credential is released only to university employees.

The one step deduction reasoning computes from the disclosure policy $\mathcal{P}_{\mathcal{D}}$ credentials $\mathcal{C}_{\mathcal{D}1} = \{C_{A5}, C_{A1}\}$. Credentials C_{A2} and C_{A3} are not considered because they are not directly disclosable

from facts in \mathcal{C}_P (the set of active credentials), which is \emptyset . Then, according to step 4 in the *PiecewiseDisclosure* algorithm, the next set of missing (and only relevant wrt $\mathcal{C}_M = \{C_{A1}, C_{A2}\}$) credentials is $\mathcal{C}_{M1} = \{C_{A1}\}$, which is requested to Alice.

In the next round, when Alice replies with certificate C_{A1} , Bob recomputes $\mathcal{C}_{D1} = \{C_{A5}, C_{A2}, C_{A3}\}$. Next, step 4 in the *PiecewiseDisclosure* algorithm selects C_{A2} as the next piecewise step of missing credentials that is then returned to Alice.

The rest of the scenario follows the same interactions as in Example 7. □

Chapter 7

System Architecture

*This chapter presents a possible architecture for interactive access control based on Business Processes for Web Services. A short introduction to Web Services is given followed by the presentation of the architecture and functional description of its components. Next, the chapter shows the implementation of the architecture using the available Web Services technologies. Then it describes the interactive access control prototype, called *iAccess*. Particularly, *iAccess* integration with the Internet security standards X.509 and SAML, and how the automated reasoning tool *DLV* is used as a back-end engine for the basic functionalities of deduction and abduction.*

7.1 A Primer on Web Services and Business Processes

A Web Service, as defined by the standard [57], is “an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service is described using a standard, formal XML notion, called its *service description*. It covers all the details necessary to interact with the service, including message formats, transport protocols and location.”

The idea behind Web services is to encapsulate and make available enterprise resources in a new heterogeneous and distributed way.

Web Services Technology Stack	
Layer	Standards
Workflow	BPEL4WS
Discovery	UDDI
Service Description	WSDL
Messaging	SOAP/XML Protocol
Transport Protocols	HTTP, HTTPS, FTP, SMTP

Figure 7.1: Web Services Technology Stack

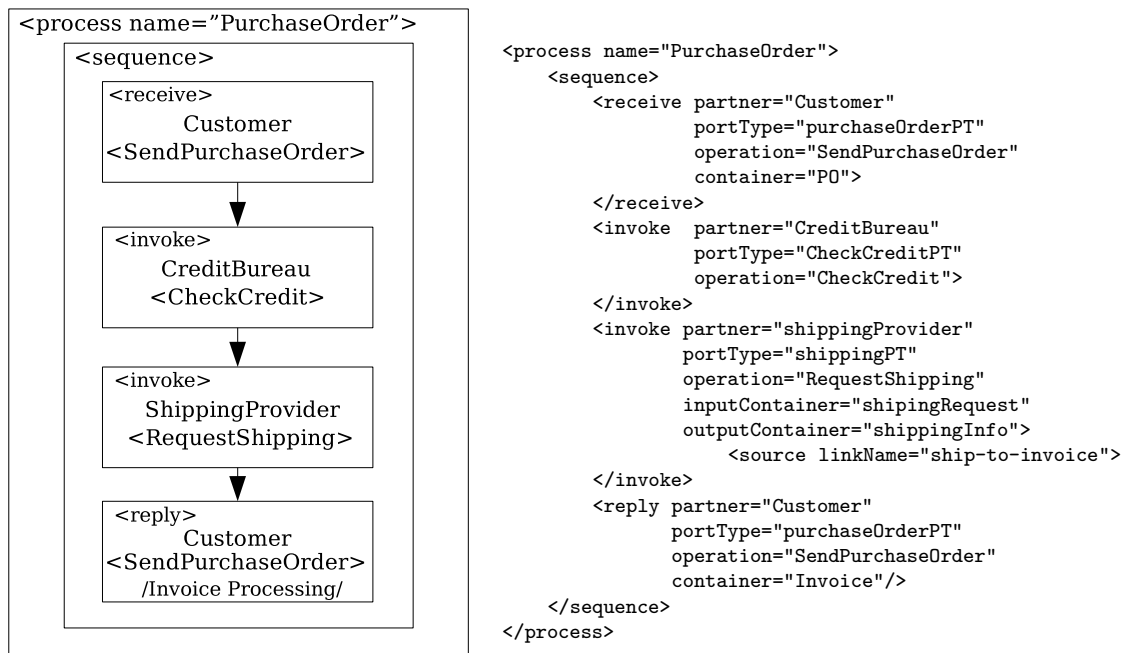


Figure 7.2: Example of a BPEL4WS Process

The WS architecture, as defined by W3C¹, is divided into five layers grouped into three main components - Wire, Description and Discovery (Figure 7.1). The *Wire* component comprises the messaging and transport layers with the SOAP protocol and the XML message format. *Discovery* offers users a unified and systematic way to find, discover and inspect service providers over the Internet. There are two standards proposed at this level - Universal Description, Discovery and Integration (UDDI) and Web Service Inspection Language (WSIL).

Moving upward we found the *Service Description* layer which is responsible for describing the basic format of offered services (protocols and encodings, where a service resides and how to invoke it). The standard for describing the communication details at this layer is Web Service Description Language (WSDL).

The *Business Process Orchestration* layer is an extension of the service model defined at the description layer. This layer is responsible for describing the behavior of complex business and workflow processes. Intuitively, business processes are graphs where each node represents an orchestration activity and primitive nodes are in WSDL. The proposed standard at this layer is the Business Process Execution Language for WS (BPEL4WS) [13].

The basic BPEL4WS primitive activities are the following:

- <invoke> invoking an operation on a Web Service.
- <receive> waiting for an operation to be invoked by someone externally.
- <reply> generating the response of an input/output operation.
- <assign> copying data from one place to another.

More complex activities can be constructed by composition:

- <sequence> allows the developer to define an ordered sequence of steps;
- <switch> allows the developer to have branching;
- <while> allows the developer to define a loop;

¹W3C Web Services Architecture: <http://www.w3.org/TR/ws-arch>.

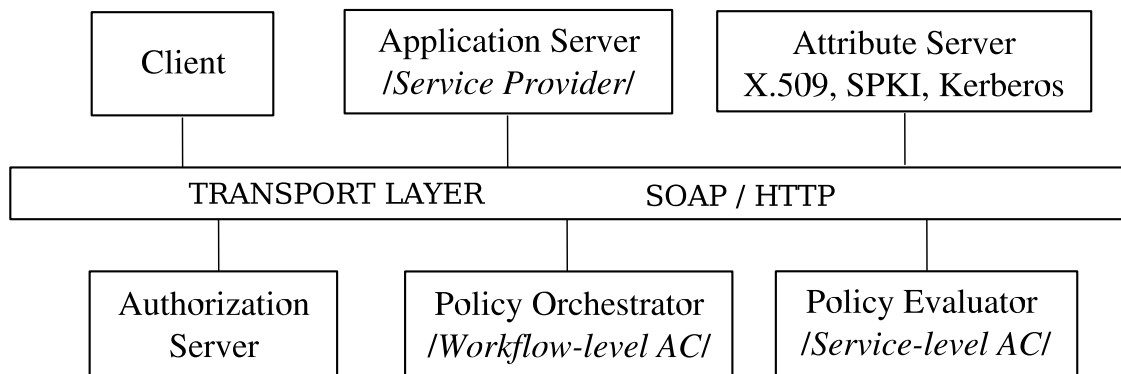


Figure 7.3: System Architecture

`<flow>` allows the developer to define that a collection of steps has to be executed in parallel.

An example of compositions of services is shown in Figure 7.2: a buyer service is ordering goods from a seller service, i.e. the buyer service invokes the order method on the seller service, whose interface is defined using WSDL. The seller service invokes a credit validation service to ensure that the buyer can pay for the goods and then continues by shipping the goods to the buyer. The credit validation service can take place at a credit bureau site in a separate security domain. Notice that a number of partners participate in the process that therefore crosses administrative boundaries.

The XML code shown in Figure 7.2 is a brief example of the scenario described above in the notations of BPEL4WS primitives. The structure of the processing section is defined by the `<sequence>` element, which states that the elements contained inside are to be executed in this order. The node content is self-explanatory.

7.2 The Architecture and Components

Combining the traditional proposals for distributed access control and the essential components used for Web Services we propose here a security architecture for orchestrating authorizations based on Business Processes for Web Services.

Figure 7.3 shows a view of the architecture. Generally, we distinguish between entities a client is aware of, the same needs to communicate with them to get a service, and entities a service provider is aware of for orchestrating (taking) an access decision. The first group is shown in the upper part of the figure and the second one at the bottom.

The Client in the system is an entity that has already discovered the service, he is interested in, and attempts to invoke it using the standard WS tools for that. Actually, from now on, we assume that all the communications between actors in the system occur as a WS request/response operations. In other words, each actor is available to other actors via a WSDL interface.

When the Client requests the ApplicationServer for invoking a service, the ApplicationServer requests the AuthorizationServer for an access decision. After the decision is taken, the ApplicationServer enforces the decision and returns the appropriate result back to the Client. The Client by its side, communicates with AttributeServers to obtain credentials that are needed to get the partner's service. Here the AttributeServer, widely termed as a certificate authority (CA), issues credentials in the form of certificates (digitally signed documents) attesting client's attributes.

We note that all of the above described entities are presented in one form or another in most of the access control systems (see §2.2) and the real contribution of the architecture, proposed in this chapter, is in the way we orchestrate and enforce authorizations using the actors PolicyEvaluator, PolicyOrchestrator and AuthorizationServer.

AttributeServer is responsible for providing group/role membership information as in [55, 64], for instance in the form of membership and non-membership certificates.

PolicyEvaluator terminology borrowed from Beznosov et al [8], is an entity responsible for achieving endpoint decisions on access control. All partners involved in a business process are likely to be as different entities, each of them represented by a **PolicyEvaluator**. It encapsulates partner's specific authorization policy and presents it as a service using WSDL.

PolicyOrchestrator from the authorization point of view is an entity responsible for the workflow level access control. It selects all partners involved in the requested service (Web service workflow) and on the base of some orchestration security policies combines the corresponding **PolicyEvaluators** in a form of a Web process, called *Policy Composition Process*, that is suitable for execution by the **AuthorizationServer**.

AuthorizationServer decouples the authorization logic from the application logic. It is responsible for *locating*, *executing* and *managing* all needed **PolicyEvaluators** and returning an appropriate result to the **ApplicationServer**. Also, it is responsible for managing all *interactions* between partners's **PolicyEvaluators** and the Client, tunneling them through the **ApplicationServer**.

At this stage one may wonder whether we need a **PolicyOrchestrator** at all. The **AuthorizationServer** could as well contact all **PolicyEvaluators** on its own. Instead, we have decided to decouple the problem in two parts: *deciding* the authorization process and *running* it. The **AuthorizationServer** runs the actual authorization process and thus queries all **PolicyEvaluators** that are needed. The entity burdened with the task of deciding what authorization process should be run is the **PolicyOrchestrator**.

We free the **AuthorizationServer** from bothering about all the details around connections between partners and **PolicyEvaluators**, as well as, **PolicyEvaluator**'s description, location, orchestration, etc. The **PolicyOrchestrator** is responsible for the *Policy Composition Service*: maintaining all relations between resources' names (services) and policies, selecting partners involved in the originally requested process and combining the corresponding **PolicyEvaluators** (as mentioned before) in a policy composition process. The **AuthorizationServer** just downloads and runs the policy composition process, as we shall see in the next sections.

7.3 Integration with Web Services Technologies

Figure 7.4 shows the entire picture with all actors and components described so far. Below we give a detailed description of how each of the entities in the figure is mapped and implemented in the final framework.

In the framework on a high level, Oracle BPEL server² (formerly Collaxa server) is used as a main BPEL4WS manager, on the **AuthorizationServer** side, for executing and managing all policy composition processes returned by the **PolicyOrchestrator** and for the implementation of the **AuthorizationServer** itself.

We chose Oracle BPEL server because:

- it supports many WS standards as BPEL4WS, WSDL, SOAP, etc;
- it interoperates with platforms as BEA's WebLogic and Microsoft .NET;
- it easily integrates Java modules (classes) within a BPEL process;

²www.oracle.com/technology/bpel

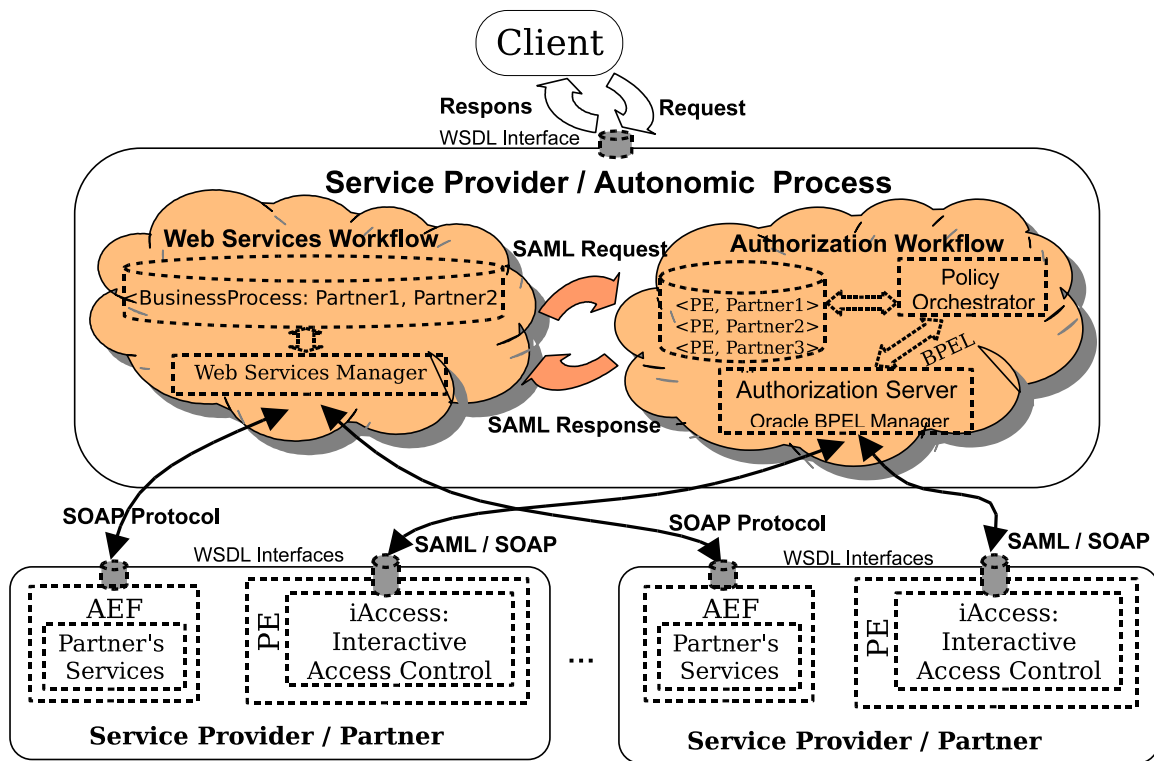


Figure 7.4: Overall View of the System Architecture and Integration

- deploying a process under the BPEL server is actually compiling it down to Java code that is internally executed by the JVM when invoked.

The AuthorizationServer itself is a BPEL process deployed and run under the BPEL manager. The AuthorizationServer, once invoked by the ApplicationServer for taking an access decision, internally deploys the policy process, returned by the PolicyOrchestrator, as an internal web service and internally executes it (i.e., starts internally the BPEL manager with the already deployed process). The advantage, in this case, is that if the AuthorizationServer is requested to get an access decision for a service that has already been asked for it and there is no change in the workflow policy then the AuthorizationServer *does not* deploy the service's policy process again but just (internally) executes it. In that way we speed up the access decision time by *JIT compilation of authorization processes*.

The PolicyOrchestrator in the current prototype is just a mapping between a service resource and its workflow policy process. We assume that the process is already created by some GUI (e.g., could be used any BPEL visual tool generator) and is available to the orchestrator. All processes returned by the PolicyOrchestrator strongly conform to the BPEL4WS specification [13].

The PolicyEvaluator (named as PE in Figure 7.4) wraps the stateless access control algorithm implementation, called *iAccess*, in a WSDL interface such that it can be uniformly accessible by the AuthorizationServer. *iAccess* prototype is described in the next section.

We have defined WSDL interfaces to all entities in the architecture so that they can communicate and discover each other using the standard for that WS tools.

An authorization example of the message flow in our architecture is the following: after the ApplicationServer has been asked for a Web Service by the Client, it requests the AuthorizationServer

to confirm whether the Client has enough access rights for that service or not. Then the **AuthorizationServer**, having client's current credentials and the service request, calls the **PolicyOrchestrator** for a policy composition process indicating what should be done for taking a decision. Once getting the policy process the **AuthorizationServer** executes it, communicating with all partners involved, and manages their interaction with the Client. When the final decision is taken (grant/deny) the **AuthorizationServer** informs the **ApplicationServer**.

Each partner in Figure 7.4 has different interfaces for communicating its access decisions and the real invocation of its target services. The first interface accepts access requests and returns access decisions using SAML protocol [44]. Once an access decision is taken it is wrapped as a SAML response and returned back to the **AuthorizationServer**. When all partners in a business process return their access decisions then these responses are kept until the real execution of the business process (ref. Web Services Workflow in Figure 7.4) takes place. Once the Web Services Manager starts executing the real process those responses are provided to each partner's application enforcement function (AEF). AEF, in turn, validates the responses and enforces the appropriate action.

7.4 *i*Access Prototype

This section describes the implementation of the interactive access control model, presented in §4.1, and its integration with the security standards X.509 and SAML.

The interactive access control model processes credentials on a high level: defines what can be inferred and what missing is from partner's access policy and user's set of credentials. There is a need of a suitable certificate infrastructure for describing participant's identities and access rights (attributes). We decided to use the widely adopted and used standard X.509 certificate framework [58].

There are two types of certificates considered by the standard: identity certificate and attribute certificate. Figure 7.5 shows the structures of the two certificates.

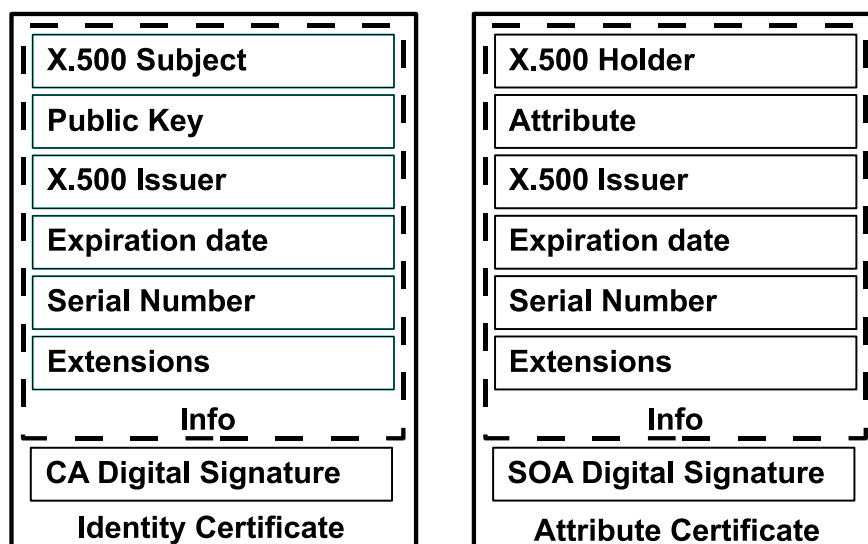


Figure 7.5: X.509 Identity and Attribute Certificates Structure

X.509 identity certificate is used to identify entities in a network. The main fields of the certificate structure are the subject information, the public key identifying the subject (corresponding to

the subject's private key), the issuer information and the digital signature on the document, signed by the issuer (by its private key).

X.509 attribute certificate has the same structure like the identity certificate with the difference that instead of public key there is a field for listing attributes and instead of subject info, it is called holder information.

Referring to the message level, we adopted to use the OASIS SAML standard [44]. OASIS SAML standard defines an XML-based framework for creating and exchanging security information between on-line partners. Thus, we use SAML for having standard semantics for authorization statements among participants in an autonomic network.

We list below the SAML Request/Response protocol and how we employ it with the interactive access control framework.

- SAML Request: use the `AuthzDecisionQuery` statement for expressing access decision requests
 - specify the `resource` and `action` in the respective standard fields of the access statement,

Once an access decision is taken we use the SAML response part:

- SAML Response: use the `AuthorizationDecisionStatement`
 - `Permit / Deny`: when explicit grant/deny is returned by the `iAccessControl` protocol (ref Figure 4.1).
 - `Indeterminate`: when `ask(CM)` is returned. We list the missing credentials in the standard SAML attribute fields, e.g.

```
<attribute name='addRequirements'>EmployeeID</attribute>
<attribute name='addRequirements'>FullProfessor</attribute>
```

So, to make the access decision engine Web Services compatible we also adopted the W3C SOAP³ as a main transport layer protocol. SOAP is a lightweight protocol for exchanging structured information in a decentralized, distributed environment. It has an optional `Header` element and a required `Body` element. Informally, in the body we specify what information is directly associated with the service request and in the header additional information that should be considered by the end-point server.

So, to request for an access decision on a message level:

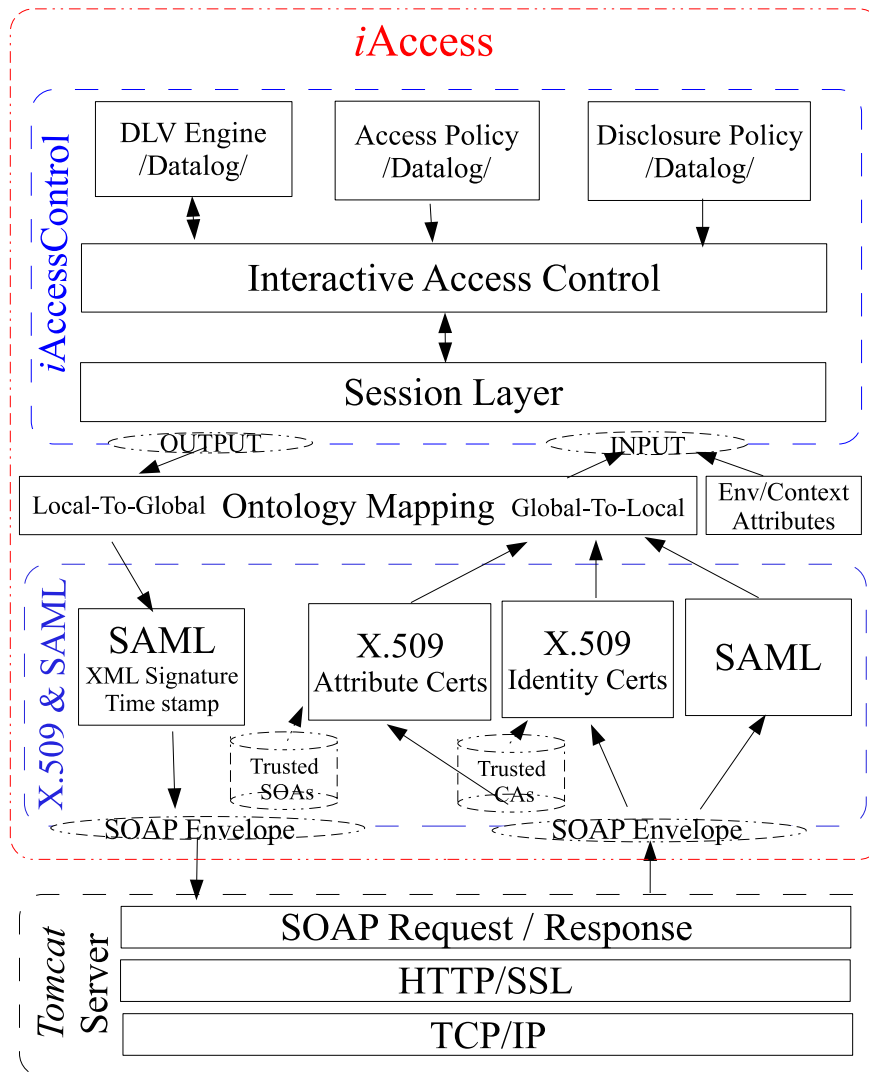
- First, place X.509 Certificates in the SOAP Header,
- Then, place OASIS SAML Request in the SOAP Body and specify it as an input to the Web Service being invoked.

Figure 7.6 shows the overall architecture of our `iAccess` prototype. The bottom most layer comprises the integration of the prototype with the Tomcat⁴ application server. We perform all requests over SSL connection. Thus, assuring message confidentiality and integrity on the transport layer.

Once an access request is received by the Tomcat server, it invokes the `iAccess` engine for an access decision. As shown in the figure, first, the engine parses the SOAP envelope, containing the body and the header elements. Then, it extracts X.509 identity and attribute certificates, and the

³<http://www.w3.org/TR/soap>

⁴<http://tomcat.apache.org>

Figure 7.6: *iAccess* Architecture

SAML request protocol. Next, the engine performs validation and verification of the certificates: first for expiration dates and second for trustworthiness. The latter is performed according to local databases listing the trusted identity issuers and, respectively, the trusted attribute issuers. The two databases are service provider specific.

Once the certificates are validated and verified, *iAccess* invokes the ontology alignment layer for mapping the global certificate information to a local, provider specific, representation. The same mapping is also performed for the SAML request protocol information.

The Ontology mapping layer transforms global-to-local and local-to-global the following information:

- certificate attributes,
- certificate issuers,
- resource names (service requests),
- actions.

These transformations are needed because on the logical level we have local (partner specific) syntax for the representation of the above items.

After the transformation is performed *iAccess* invokes the *iAccessControl* module for an access decision. The *iAccessControl* module transforms certificates' information and SAML request to predicates suitable for the logical model. We have done the following transformations:

- identity certificates are transformed to `certificate(subject, Issuer:i)` predicates,
- attribute certificates are transformed to `credential(holder, Attr:a, Issuer:i)` predicates,
- SAML access request to `grant(Resource:s, Action:p)`.

Once an access decision is taken (by the *iAccessControl* protocol) *iAccess* maps the information grant, deny or additional credentials to their global representation and then generates the respective SAML Response protocol. After that, *iAccess* places a time-stamp for validity period on the access decision statement and then digitally signs it to ensure integrity of the information. Next, Tomcat server returns the SAML decision to the entity requested it.

However, we have not seen yet how to cope with the implementation of the interactive access control algorithms. Next section describes how one can employ the DLV reasoning tool to perform the basic computations of abduction and deduction.

7.5 Integration with Automated Reasoning Tool DLV

For the implementation of the stateless access control algorithm, introduced in Chapter 4, we use the DLV⁵ system (a disjunctive datalog system with negations and constraints) as a core engine for the basic functionalities of deduction and abduction. The disjunctive datalog front-end (the default one) is used for deductive computations while the diagnosis front-end is used for abductive computations.

Figure 7.7 shows the implementation using the DLV system. The input is the requested service r , the policy for access control \mathcal{P}_A , the policy for disclosure control \mathcal{P}_D , the set of active credentials \mathcal{C}_P and the set of declined credentials \mathcal{C}_N . Step 1 uses the DLV's deductive front-end. It specifies as input the service request r marked as a query over the models computed from the second argument: \mathcal{P}_A and \mathcal{C}_P . The output of this step are those models of $\mathcal{P}_A \cup \mathcal{C}_P$ in which r is true.

If it exists a model in Step 1 that satisfies r then it is returned *grant* (step 1). If no model for r exists (step 2) then we use the DLV's deductive front-end with input partner's disclosure policy \mathcal{P}_D together with \mathcal{C}_P . In this case, DLV computes all credentials disclosable from $\mathcal{P}_D \cup \mathcal{C}_P$. Then from the computed set we remove all credentials that belong to \mathcal{C}_N and \mathcal{C}_P .

Once the disclosable credentials are computed then, in Step 3, we use the abductive diagnosis front-end with the following input: the requested service r , stored in a temporary file with extension *.obs* (observations), the just computed set of disclosable credentials \mathcal{C}_D stored in a temporary file with extension *.hyp* (hypotheses or also called abducibles) and the third argument is the access policy together with the active credentials, $\mathcal{P}_A \cup \mathcal{C}_P$. The two input files (*.hyp* and *.obs*) are particular for the DLV in the abductive mode.

The output of that computation are all possible subsets of the hypotheses that satisfy the observations. In that way we find all possible missing sets of credentials satisfying r . Then we filter them first against role-minimality criterion and then against set cardinality criterion. The former filters those sets with lowest possible role-position values and the latter filters the ones with minimal cardinality.

If no missing set is found then we return deny else we return the missing set back to the client.

⁵www.dlvsystem.com

```

iAccessDecision( $r, \mathcal{P}_A, \mathcal{P}_D, \mathcal{C}_P, \mathcal{C}_N$ ){
  1: if doDeduction( $r, \mathcal{P}_A \cup \mathcal{C}_P$ ) then return grant
  2: else  $\mathcal{C}_D = \{\text{run DLV in deduction mode with input: } \mathcal{P}_D \cup \mathcal{C}_P\} \setminus (\mathcal{C}_N \cup \mathcal{C}_P)$ ;
  3:  $result = \text{doAbduction}(r, \mathcal{C}_D, \mathcal{P}_A \cup \mathcal{C}_P)$ ;
  4: if  $result == \perp$  then return deny
  5: else  $\mathcal{C}_M = result$  and return  $ask(\mathcal{C}_M)$ ;
}

doDeduction( $R$ : Query,  $P$ : LogProgram){ // check for  $P \models R$ ?
  1: run DLV in deduction mode with input:  $P, R$ ? ;
  2: check output: if  $R$  is deducible then return true else return false;
}

doAbduction( $R$ : Observation,  $H$ : Hypotheses,  $P$ : LogProgram){
  1: run DLV in abduction diagnosis mode with input:  $R, H, P$  ;
  2: DLV output: all sets  $C_i$  that (i)  $C_i \subseteq H$ , (ii)  $P \cup C_i \models R$ , (iii)  $P \cup C_i \not\models \perp$ ;
  3: if no  $C_i$  exists then return  $\perp$ 
  4: else select a minimal  $C_{min}$  and return  $C_{min}$ ;
}

```

Figure 7.7: Implementation of the Basic Functionalities of Deduction and Abduction

Remark 5 *The sequence, the two criteria, set cardinality and role minimality makes sense in different contexts. The sequence role minimality/set cardinality, tries to keep the minimal set as lower in the role hierarchy as possible, i.e. selects those sets that have a larger number of not so powerful roles. The other alternative, set cardinality/role minimality, selects those sets with fewer roles but with higher privileges.*

The latter may be useful if getting or transmitting credentials is expensive, e.g. in a mobile setting.

Chapter 8

Conclusions

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

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 have presented the key interactive access control algorithm that, in case service request fails, computes on the fly missing credentials that entail the request. Following that, we enriched the framework over the existing policy-based approaches by introducing the difference between *disclosable* and *hidden* credentials and between *monotonic* and *well-behaved* policies. The first distinction addresses the behavior of an autonomic server allowing it to dynamically protect the privacy of his policies by specifying which credentials are hidden and which are not. This allows a server to restrict access to certain services only to selected clients. The latter distinction extends our work on a wider set of policy languages wrt the already existing approaches [12, 63, 37].

Further, we have extended the algorithm to cope with arbitrary access policies so that in cases of inconsistency it performs a recovery step and finds a set of excessing credentials banning the client to get a solution for the desired resource. We have presented a strengthened version of the algorithm resistant to DoS attacks. We have also proved that for cooperative and compliant clients the basic and extended access control frameworks are complete and correct, i.e., a client with the right credentials for a request will get access and if access is granted then the client has the right set of credentials.

We have identified the interactive access control model as a way for protecting security interests wrt disclosure of information and access control of both server and client sides. We have proposed a protocol for leveraging trust negotiation between two entities involved in an autonomic communication. The protocol communicates and negotiates the missing credentials in a piecewise manner until enough trust is established and the service is granted or the negotiation fails and the process is terminated. The protocol can be run on both client and server sides so that they understand each other and automatically interoperate until a desired solution is reached or denied.

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, performed on the policies, do not require any specific policy structure.

We have also presented an implementation of the framework using X.509 and SAML standards. Web Services technology (WSDL, BPEL and SAML) is used for unifying and managing partners' requirements and communications on a high level in the architecture. DLV system is used as a back-end engine for the basic functionalities of deduction and abduction.

8.1 Open Problems and Future Work

Future work is in the direction of characterizing the complexity of the framework. General abduction lays at the second level of the polynomial hierarchy but our problems are at the same time more specialized (e.g. credentials are occurring only positively in the rules) and more general (we have hierarchies of roles so subset or cardinality minimality does not really apply).

Performing experimental assessments of *iAccess* prototype on real industrial case studies. Proving which guarantees the protocol can offer (e.g., in terms of interoperability, completeness and correctness) when applied to specific policy languages is still an open process and will also be a subject of future research.

In the direction of mutual negotiation, we want to explore the interoperability of the negotiation framework with the TrustBuilder prototype described by Yu et al. [63]. We believe that this is an important step toward building a secure open computing environment.

Bibliography

- [1] ALTENSCHMIDT, C., BISKUP, J., FLEGEL, U., AND KARABULUT, Y. Secure mediation: Requirements, design, and architecture. *Journal of Computer Security* 11, 3 (2003), 365–398.
- [2] APT, K. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier, 1990.
- [3] ATLURI, V., CHUN, S. A., AND MAZZOLENI, P. A Chinese wall security model for decentralized workflow systems. In *Proceedings of the 8th ACM conference on Computer and Communications Security* (2001), ACM Press, pp. 48–57.
- [4] AU, R., LOOI, M., AND ASHLEY, P. Cross-domain one-shot authorization using smart cards. In *Proceedings of the 7th ACM conference on Computer and communications security* (2000), ACM Press, pp. 220–227.
- [5] BACON, J., AND MOODY, K. Toward open, secure, widely distributed services. *Communications of the ACM* 45, 6 (2002), 59–64.
- [6] BERTINO, E., CATANIA, B., FERRARI, E., AND PERLASCA, P. A logical framework for reasoning about access control models. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies (SACMAT)* (2001), ACM Press, pp. 41–52.
- [7] BERTINO, E., FERRARI, E., AND ATLURI, V. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security (TISSEC)* 2, 1 (1999), 65–104.
- [8] BEZNOSOV, K., DENG, Y., BLAKLEY, B., BURT, C., AND BARKLEY, J. A resource access decision service for CORBA-based distributed systems. In *Proceedings of 15th IEEE Annual Computer Security Applications Conference. (ACSAC '99)* (1999), IEEE Press, pp. 310–319.
- [9] BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J., AND KEROMYTIS, A. D. The role of trust management in distributed systems security. In *Secure Internet programming: security issues for mobile and distributed objects*. Springer-Verlag, 1999, pp. 185–210.
- [10] BLAZE, M., FEIGENBAUM, J., AND KEROMYTIS, A. D. KeyNote: Trust Management for Public-Key Infrastructures. In *Proceedings of 6th International Workshop on Security Protocols* (1998), vol. 1550 of *Lecture Notes in Computer Science*, pp. 59–63.
- [11] BLAZE, M., FEIGENBAUM, J., AND LACY, J. Decentralized trust management. In *Proceedings of IEEE Symposium on Security and Privacy* (1996), IEEE Press, pp. 164–173.
- [12] BONATTI, P., AND SAMARATI, P. A unified framework for regulating access and information release on the web. *Journal of Computer Security* 10, 3 (2002), 241–272.

- [13] BPEL4WS. Business Process Execution Language for Web Services (BPEL4WS), 2003. <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel>.
- [14] CHADWICK, D., OTENKO, A., AND BALL, E. Role-based access control with X.509 attribute certificates. *IEEE Internet Computing* 7, 2 (Mar/Apr 2003), 62–69.
- [15] CHADWICK, D. W., AND OTENKO, A. The PERMIS X.509 role-based privilege management infrastructure. In *Seventh ACM Symposium on Access Control Models and Technologies* (2002), ACM Press, pp. 135–140.
- [16] CHU, Y.-H., FEIGENBAUM, J., LAMACCHIA, B., RESNICK, P., AND STRAUSS, M. REF-EREE: Trust management for Web applications. *Computer Networks and ISDN Systems* 29, 8–13 (1997), 953–964.
- [17] DAMIANOU, N., DULAY, N., LUPU, E., AND SLOMAN, M. The Ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY)* (January 2001), Springer-Verlag, pp. 18–38.
- [18] DE CAPITANI DI VIMERCATI, S., AND SAMARATI, P. Access control: Policies, models, and mechanism. In *Foundations of Security Analysis and Design - Tutorial Lectures*, R. Focardi and F. Gorrieri, Eds., vol. 2171 of *LNCS*. Springer-Verlag Press, 2001.
- [19] ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B. M., AND YLONEN, T. *SPKI Certificate Theory*, September 1999. IETF RFC 2693.
- [20] FERRAILOLO, D. F., SANDHU, R., GAVRILA, S., KUHN, D. R., AND CHANDRAMOULI, R. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)* 4, 3 (2001), 224–274.
- [21] GELFOND, M., AND LIFSCHITZ, V. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming (ICLP'88)* (1988), R. Kowalski and K. Bowen, Eds., MIT-Press, pp. 1070–1080.
- [22] GEORGAKOPOULOS, D., HORNICK, M. F., AND SHETH, A. P. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases* 3, 2 (April 1995), 119–153.
- [23] HINE, J. A., YAO, W., BACON, J., AND MOODY, K. An architecture for distributed OASIS services. In *IFIP/ACM International Conference on Distributed systems platforms* (2000), Springer-Verlag New York, Inc., pp. 104–120.
- [24] JOHNSTON, W., MUDUMBAL, S., AND THOMPSON, M. Authorization and attribute certificates for widely distributed access control. In *Proceedings of Seventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '98)* (1998), IEEE Press, pp. 340–345.
- [25] KANG, M. H., PARK, J. S., AND FROSCHE, J. N. Access control mechanisms for inter-organizational workflow. In *Proceedings of the Sixth ACM Symposium on Access control models and technologies* (2001), ACM Press, pp. 66–74.
- [26] KOSHUTANSKI, H. A survey on distributed access control systems for web business processes. *International Journal of Network Security (IJNS)* (To appear).

- [27] KOSHUTANSKI, H., AND MASSACCI, F. An access control framework for business processes for Web services. In *Proceedings of the 2003 ACM workshop on XML security* (Fairfax, VA, October 2003), ACM Press, pp. 15–24.
- [28] KOSHUTANSKI, H., AND MASSACCI, F. An access control system for business processes for Web services. Tech. rep., Nordic Workshop on Secure IT Systems (NORDSEC), Gjøvik University College, Norway, October 2003.
- [29] KOSHUTANSKI, H., AND MASSACCI, F. A logical model for security of Web services. Tech. Rep. IIT TR-10/2003, First International Workshop on Formal Aspects in Security and Trust (FAST), Istituto di Informatica e Telematica, Pisa, Italy, September 2003. Editors: Theo Dimitrakos and Fabio Martinelli.
- [30] KOSHUTANSKI, H., AND MASSACCI, F. E pluribus unum: Deduction, abduction and induction, the reasoning services for access control in autonomic communication. In *Proceedings of the 1st IFIP TC6 WG6.6 International Workshop on Autonomic Communication (WAC)* (Berlin, Germany, October 2004), vol. 3457 of *LNCS*, Springer-Verlag Press, pp. 179–190.
- [31] KOSHUTANSKI, H., AND MASSACCI, F. Interactive access control for Web Services. In *Proceedings of the 19th IFIP Information Security Conference (SEC 2004)* (Toulouse, France, August 2004), Kluwer Press, pp. 151–166.
- [32] KOSHUTANSKI, H., AND 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, August 2004), Kluwer Press, pp. 139–152.
- [33] KOSHUTANSKI, H., AND MASSACCI, F. A system for interactive authorization for Business Processes for Web Services. In *Proceedings of the 4th International Conference on Web Engineering (ICWE 2004)* (Munich, Germany, July 2004), Springer-Verlag Press, pp. 521–525.
- [34] KOSHUTANSKI, H., AND MASSACCI, F. Interactive credential negotiation for stateful business processes. In *Proceedings of the 3rd International Conference on Trust Management (iTrust)* (Rocquencourt, France, May 2005), vol. 3477 of *LNCS*, Springer-Verlag Press, pp. 257–273.
- [35] LEONE, N., PFEIFER, G., AND ET AL. The DLV system. In *the 8th European Conference on Artificial Intelligence (JELIA)* (September 2002), vol. 2424 of *Lecture Notes in Computer Science*, Springer-Verlag Press, pp. 537–540.
- [36] LI, N., GROSOFF, B. N., AND FEIGENBAUM, J. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC)* 6, 1 (2003), 128–171.
- [37] LI, N., AND MITCHELL, J. C. RT: A role-based trust-management framework. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX III)* (Los Alamitos, California, April 2003), IEEE press, pp. 201–212.
- [38] LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. Design of a role-based trust-management framework. In *Proceedings of IEEE Symposium on Security and Privacy, 2002. S&P (2002)*, IEEE Press.
- [39] LI, N., WINSBOROUGH, W. H., AND MITCHELL, J. C. Distributed credential chain discovery in trust management. *Journal of Computer Security* 11, 1 (February 2003), 35–86.

- [40] LYMBEROPOULOS, L., LUPU, E., AND SLOMAN, M. An adaptive policy based framework for network services management. *Plenum Press Journal of Network and Systems Management* 11, 3 (September 2003), 277–303.
- [41] NIEMELÄ, I., SIMONS, P., AND SOININEN, T. Stable model semantics of weight constraint rules. In *Proceedings of the Fifth International Conference on Logic Programming and Non-monotonic Reasoning* (December 1999), Springer-Verlag Press.
- [42] OPPLIGER, R., GREULICH, A., AND TRACHSEL, P. A distributed certificate management system (DCMS) supporting group-based access controls. In *Proceedings of 15th IEEE Annual Computer Security Applications Conference. (ACSAC '99)* (1999), IEEE Press, pp. 241–248.
- [43] PARK, J. S., AND SANDHU, R. RBAC on the Web by smart certificates. In *Proceedings of the fourth ACM workshop on Role-based access control* (1999), ACM Press, pp. 1–9.
- [44] SAML. Security Assertion Markup Language (SAML), 2004. www.oasis-open.org/committees/security.
- [45] SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. Role-based access control models. *IEEE Computer* 39, 2 (February 1996), 38–47.
- [46] SEAMONS, K., AND WINSBOROUGH, W. Automated trust negotiation. Tech. rep., US Patent and Trademark Office, 2002. IBM Corporation, patent application filed March 7, 2000.
- [47] SHANAHAN, M. Prediction is deduction but explanation is abduction. In *Proceedings of IJCAI '89* (1989), Morgan Kaufmann, pp. 1055–1060.
- [48] SLOMAN, M., AND LUPU, E. Policy specification for programmable networks. In *Proceedings of the First International Working Conference on Active Networks* (1999), Springer-Verlag, pp. 73–84.
- [49] SMIRNOV, M. Rule-based systems security model. In *Proceedings of the Second International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS)* (2003), Springer-Verlag Press, pp. 135–146.
- [50] SPKI. SPKI certificate theory, 1999. IETF RFC 2693.
- [51] THOMPSON, M., JOHNSTON, W., MUDUMBAI, S., HOO, G., JACKSON, K., AND ESSIARI, A. Certificate-based access control for widely distributed resources. In *Proceedings of Eighth USENIX Security Symposium (Security'99)* (August 1999), pp. 215–228.
- [52] VARADHARAJAN, V., CRALL, C., AND PATO, J. Authorization in enterprise-wide distributed system: a practical design and application. In *Proceedings of 14th IEEE Annual Computer Security Applications Conference* (1998), IEEE Press, pp. 178–189.
- [53] VARADHARAJAN, V., CRALL, C., AND PATO, J. Issues in the design of secure authorization service for distributed applications. In *Global Telecommunications Conference. GLOBECOM 1998. The Bridge to Global Integration* (1998), vol. 2, IEEE Press, pp. 874–879.
- [54] WEEKS, S. Understanding trust management systems. In *IEEE Symposium on Security and Privacy (SS&P)* (2001), IEEE Press.
- [55] WOO, T. Y. C., AND LAM, S. Designing a distributed authorization service. In *Proceedings of Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. INFOCOM* (1998), vol. 2, IEEE Press, pp. 419–429.

- [56] WOO, T. Y. C., AND LAM, S. S. A framework for distributed authorization. In *Proceedings of the 1st ACM conference on Computer and communications security* (1993), ACM Press, pp. 112–118.
- [57] WS CONCEPTUALARCHITECTURE. *Web Services Conceptual Architecture (WSCA 1.0)*, May 2001. <http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>.
- [58] X.509. The directory: Public-key and attribute certificate frameworks, 2001. ITU-T Recommendation X.509:2000(E) | ISO/IEC 9594-8:2001(E).
- [59] XACML. eXtensible Access Control Markup Language (XACML), 2004. www.oasis-open.org/committees/xacml.
- [60] YAO, W. Fidelis: A policy-driven trust management framework. In *iTrust* (2003), P. Nixon and S. Terzis, Eds., vol. 2692 of *Lecture Notes in Computer Science*, Springer-Verlag Press, pp. 301–317.
- [61] YAO, W. *Trust management for widely distributed systems*. PhD thesis, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, 2004. Appeared as a technical report UCAM-CL-TR-608, ISSN 1476-2986.
- [62] YU, T., AND WINSLETT, M. A unified scheme for resource protection in automated trust negotiation. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2003), IEEE press, pp. 110–122.
- [63] YU, T., WINSLETT, M., AND SEAMONS, K. E. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security (TISSEC)* 6, 1 (2003), 1–42.
- [64] ZURKO, M., SIMON, R., AND SANFILIPPO, T. A user-centered, modular authorization service built on an RBAC foundation. In *Proceedings of the IEEE Symposium on Security and Privacy* (1999), IEEE Press, pp. 57–71.