

Expressive and Deployable Access Control in Open Web Service Applications

C.A. Ardagna, S. De Capitani di Vimercati, S. Paraboschi, E. Pedrini, P. Samarati, M. Verdicchio

Abstract—Traditional access control solutions, based on preliminary identification and authentication of the access requester, are not adequate for the context of open Web service systems, where servers generally do not have prior knowledge of the requesters. The research community has acknowledged such a paradigm shift and several investigations have been carried out for new approaches to regulate access control in open dynamic settings. Typically based on logic, such approaches, while appealing for their expressiveness, result not applicable in practice, where simplicity, efficiency, and consistency with consolidated technology are crucial. The eXtensible Access Control Markup Language (XACML) has established itself as the emerging technological solution for controlling access in an interoperable and flexible way. Although supporting the most common policy representation mechanisms and having acquired a significant spread in the research community and the industry, XACML still suffers from some limitations which impact its ability to support actual requirements of open Web-based systems. In this paper, we provide a simple and effective formalization of novel concepts that have to be supported for enforcing the new access control paradigm needed in open scenarios, toward the aim of providing an expressive solution actually deployable with today's technology. We illustrate how the concepts of our model can be deployed in the XACML standard by exploiting its extension points for the definition of new functions, and introducing a dialog management framework to enable access control interactions between Web service clients and servers.

Index Terms—Deployable Access Control, Web Services, Credentials, Security Policy Communication, XACML.

1 INTRODUCTION

ACCESS control is the component of security systems responsible to evaluate if a subject can be allowed to operate in a given way on a specific resource. In traditional centralized environments, the focus has been usually directed to the design of solutions able to provide efficient decisions on whether an access request, usually represented as a triple (subject,object,action), should be permitted or refused. Security requirements impose that the control has to be realized for every access to a resource and even a modest increase in the evaluation time can have a great impact on general system performance. As a consequence, low-level access control models with limited flexibility have been used. Another aspect characterizing most traditional applications is the limited variety of users. Users could be adequately classified by their identifiers, assigned by the system and presented by the users at every access, and by the group (or role) characterizing them.

Web service-based systems, in which servers receive and process requests from remote parties, are characterized by features that force a complete revision of the solutions adopted so far. First of all, these systems are designed to be open, which implies that the servers may not have preliminary knowledge of the requester. Such a circumstance makes traditional access control

inapplicable, as it is based first on preliminary identification and authentication of the requester, and then on checks against applicable authorizations. Rather, the Web server needs to determine the conditions under which access can be granted, and to communicate them to the requester. Moreover, Web services are supposed to be platform-independent, which triggers the need for the system behavior to be specified by means of a general description language (typically, XML-based) that does not rely on any assumption on what language the services are implemented with, nor on what protocol is used for message exchange over the communication channels.

The paradigm shift from requester identification to access condition communication and the need for a general system behavior description have already been acknowledged by the research community. Many works and progresses in credential-based and attribute-based access control rely on the idea that the server communicates to the requester the credentials that she must possess, or the properties that she needs to satisfy, to acquire access. Several works have also investigated the different aspects of credential-based access control, and presented different models and languages. Typically, a logic-based language is proposed, allowing for compact and expressive specifications of the access control policy as well as its communication to the requester. Furthermore, when privacy is an issue, these works assume that the requester can enforce her own policy and initiate a negotiation with the server, during which access policies, credentials, and attributes are exchanged, until access is eventually granted or denied. Nevertheless, many logic-based proposals, while appealing for their expressive-

- C.A. Ardagna, S. De Capitani di Vimercati, E. Pedrini, P. Samarati are with Department of Information Technology, Università degli Studi di Milano, 26013 Crema, Italy. E-mail: {claudio.ardagna, sabrina.decapitani, eros.pedrini, pierangela.samarati}@unimi.it — S. Paraboschi and M. Verdicchio are with Dipartimento di Ingegneria dell'Informazione e Metodi Matematici of the Università degli Studi di Bergamo, 24044 Dalmine, Italy. E-mail: {parabosc, mario.verdicchio}@unibg.it

ness, turn out to be not applicable in practice, where simplicity, efficiency, and consistency with consolidated technology are crucial.

Effectively tackling these issues is probably the key to the success of the eXtensible Access Control Markup Language (XACML) [1]. XACML is an OASIS standard that proposes an XML-based language for specifying and exchanging access control policies over the Web. The language can support the most common security policy representation mechanisms and has already found significant support by many players. Moreover, it includes standard extension points for the definition of new functions, data types, and policy combination methods, which provide a great potential for the management of access control requirements in future environments.

We aim at realizing such a potential, especially with respect to those aspects of open world scenarios that are not supported by the current XACML standard. In particular, while providing a flexible and extensible attribute-based access control, XACML does not include features to deal with and reason about credentials. It also assumes the requester to be known, or to provide all her attributes at request time, thus enforcing access control with a traditional closed world setting. In this paper, we propose an extension to XACML that enables clients and servers to rely also on credentials to request access and specify policies, respectively. The additional reasoning capabilities needed by the architecture components to take credentials into account will be designed in a way that does not clash with the efficiency restrictions met by the original XACML.

The contribution of this paper is threefold. First, starting from an analysis of the shortcomings of XACML, we provide a simple formalization of concepts that should be expressed to effectively support real world applications. Our formalization includes a novel proposal for expressing the different ways a server can present its access control policy to a requester. These different ways, characterized as disclosure policies, are specified at the fine grain of conditions appearing in a policy specification, thus allowing flexibility in dialog management. Second, we illustrate how such concepts can be deployed in XACML, exploiting XACML extensibility, introducing new elements, attributes, or XACML profiles. Third, we pinpoint the specific changes needed in the XACML architecture and implementation to incorporate our extensions. Although explicitly referring to XACML, our proposal must not be considered simply as an improvement of XACML. Instead, the approach we present can be viewed as a proposal for the definition of a general policy language able to support important requirements of novel Web service applications. The basic principles guiding our effort are first illustrated on a standard-independent, abstract level. Then, to demonstrate that our ideas can find a direct application within current systems, we focus our attention on how XACML can be adapted and extended to support these new features.

The remainder of this paper is organized as follows.

Section 2 illustrates the novel features that should be supported by a practical access control in open Web service applications. Section 3 presents the formal modeling of the novel features that we have identified. Section 4 illustrates how such formal concepts can be effectively deployed in XACML. Section 5 presents enhancements to be made to the standard XACML architecture to support the extended XACML language. Section 6 discusses related work. Finally, Section 7 gives our concluding remarks.

2 FEATURES TO BE SUPPORTED FOR PRACTICAL ACCESS CONTROL

Our analysis aims at identifying the novel concepts that access control in open Web service applications should support. The analysis takes as a reference XACML, which is well recognized as the standard access control language for Web applications, and identifies a few limitations in using XACML in this scenario.

- *Certified information.* The XACML policies allow the definition of generic boolean conditions for referring to the different elements (e.g., subject, object, action) of a policy. It is possible to define a generic subject expression that allows referring to a set of subjects satisfying the specified conditions. XACML therefore enforces attribute-based access control, since the authorizations applicable to a requester depend on the properties the requester presents (and the conditions that the requester satisfies). The properties are assumed to be known during the evaluation time and stored in the XACML evaluation context, or presented by the requester together with the request. XACML also permits expressing the fact that some properties should be certified; this is limited however to conditions on attributes *issuer*, *time*, and *date* that can be associated with the property. While XACML acknowledges that properties can be presented by means of certificates, and as a matter of fact, it has been designed to be integrated with the Security Assertion Markup Language (SAML) [2] for exchanging various types of security assertions and for providing protocol mechanisms, it does not provide a real support for expressing and reasoning about digital certificates in the specification of the authorization policies. Intuitively, XACML supports attribute-based access control but does not really support credential-based access control. While the two concepts are often used in an indistinguishable way in the literature, the difference is noticeable in XACML. Credential-based access control needs support for expressing that some properties should be presented by means of given certificates, possibly imposing conditions, besides on values of the properties, also on the certificates themselves (e.g., their type or issuer).
- *Abstractions.* Abstractions have been considered in several novel logic-based proposals (typically re-

ferred to as abbreviations, macros, or ontological reasoning) and allow the derivation of new concepts based on others. Intuitively, abstractions represent a shorthand for expressing, with a single concept, a more complex definition (e.g., a set, a disjunction, or a conjunction of concepts). Use of abstractions in the policy specification provides therefore a compact and easy way to refer to complex concepts. For instance, `id_document` can be defined as an abstraction for any element in the set of credentials `{identity_card, driver_license, passport}`. An authorization specifying that the requester needs to provide an `id_document` for accessing a specific Web service can then be satisfied by the requester presenting any of the three credentials above. XACML does not provide explicit support for abstractions.

- *Recursion.* While recursive reasoning on data can be captured by the application, recursive reasoning on credentials or their properties needs to be captured by the authorization specification language. Recursive reasoning is in fact needed for expressing policies based on chains of credentials and for supporting delegation. XACML does not provide any support for recursive reasoning.
- *Dialog Management.* XACML implicitly assumes that the engine enforcing access control has available all information needed to evaluate whether an authorization policy is satisfied, without the need of an infrastructure to manage the dialog between parties. The evaluation of a policy can result therefore in four possibilities: permit, deny, not applicable (if the policy does not apply to the request), or indeterminate (if the server does not have the information necessary to evaluate the policy). In an open world scenario, the approach above would require the requester to reveal all the necessary credentials together with the service request. Assuming that the requester can hand over all her credentials is simply inconceivable: the requester will want the ability to send to the counterpart only just what is needed to acquire access. On the other hand, the requester cannot be assumed to know the credentials that are needed for an access. The server should therefore support a new way of enforcing the access control process, which cannot be assumed anymore to operate with a given prior knowledge and return a definite access decision. Rather, the access control process should be able to operate without a-priori knowledge of the requester and should return the conditions that it requires be satisfied for the access to be allowed [3]. Extending XACML with dialog management would then avoid the simple evaluation to indeterminate of all those cases for which the server is missing information, providing instead the possibility to communicate to the requester which information is required and therefore enabling the requester to provide it and acquire access to the web service.

3 MODELING NOVEL CONCEPTS

We present a formal description of the novel concepts discussed in the previous section. Our goal is not the definition of a complete authorization model, but the formal representation of the basic building blocks needed for referring to credentials and reasoning about them, and for supporting abstractions, recursion, and dialog.

3.1 Credentials

Credential support requires the possibility of explicitly referring to digital certificates and relevant conditions about them in the policy specifications. We formally model a credential as follows.

Definition 3.1 (Credential): A credential is represented by a symbol c and is modeled as a pair $(\mathcal{M}_c, \mathcal{A}_c)$, where:

- \mathcal{M}_c is a list of metadata name-value pairs $(\langle M_1, m_1 \rangle, \dots, \langle M_k, m_k \rangle)$ that represent properties on the credential (e.g., $\langle \text{type}, \text{id_card} \rangle$ is a metadata describing a credential of type `id_card`);
- \mathcal{A}_c is a list of attribute name-value pairs $(\langle A_1, a_1 \rangle, \dots, \langle A_n, a_n \rangle)$ that represent the content of the credential (e.g., $\langle \text{last_name}, \text{Smith} \rangle$ represents attribute `last_name` whose value is `Smith`).

The *schema* of a credential c , denoted \underline{c} , is the set of its metadata and attribute names alone, without the specific instances of values. Formally, $\underline{c} = (\underline{\mathcal{M}}_c, \underline{\mathcal{A}}_c)$, where $\underline{\mathcal{M}}_c = (M_1, \dots, M_k)$, and $\underline{\mathcal{A}}_c = (A_1, \dots, A_n)$. Our notation relies on the triangle (\triangleright) symbol and the dot (\cdot) symbol to create *credential terms* that refer to metadata and attribute names, respectively, in a credential schema. For instance, terms $c \triangleright M$ and $c.A$ refer to metadata name M and to attribute name A in credential c , respectively.

Credential terms can then be used to specify *credential conditions* on certified properties and on the certificates themselves, which can be used like any other condition within a subject expression (i.e., the attribute-based expression identifying subjects to which the authorization applies).

Definition 3.2 (Simple credential condition): A simple credential condition is either:

- a credential term t , where t is $c \triangleright M$ or $c.A$, with $\underline{c} = (\underline{\mathcal{M}}_c, \underline{\mathcal{A}}_c)$, $M \in \underline{\mathcal{M}}_c$, and $A \in \underline{\mathcal{A}}_c$;
- an expression of the form $t \pi v$, where t is credential term, π is a symbol representing a standard predicate (e.g., $'='$, $'\neq'$, $'>'$, $'\in'$), and v is a metadata value, an attribute value, or another credential term.

Conditions of the first type (credential terms) have the semantics of requiring the requester to hold a credential with the specified term. If no further condition is specified on the corresponding metadata/attribute in the subject expression, the semantics is simply that the metadata/attribute needs to be presented, although its value does not impact the access control decision. For instance, a subject expression can include a term $c.\text{last_name}$ as a condition, without any further restriction on attribute

last_name; while the value of the attribute is not taken into consideration in the access control decision process, it might be needed for logging purposes (as the name of those who have accessed a service is logged).

Conditions of the second type are satisfied if the requester holds a credential with a term that satisfies the stated restriction. For instance, $c.last_name = \text{Smith}$ is satisfied by a credential including attribute *last_name* whose value is Smith. Simple credential conditions can be combined by using the AND and OR boolean operators to create more complex conditions, as formally defined as follows.

Definition 3.3 (Credential condition): A credential condition is inductively defined as:

- a simple credential condition, or
- $s_1 \wedge s_2$, or
- $s_1 \vee s_2$,

where s_1 and s_2 are credential conditions.

A credential condition represents the basic construct to be used in the definition of a subject expression in a policy that permits referring to a set of subjects satisfying certain conditions. Note that a credential condition may include multiple occurrences of credential symbols. If the same symbol is used, the corresponding simple conditions refer to (and must therefore be satisfied by) the same credential; if a different symbol is used, the corresponding simple conditions can be satisfied by different credentials or must be satisfied by different credentials, if an inequality between the credential symbols is explicitly specified in the credential condition. For instance, condition " $c_1 \triangleright type = \text{passport} \wedge c_1.last_name = \text{Smith} \wedge c_1.nationality = \text{US}$ " can be satisfied by presenting a credential of *type* *passport* containing attributes *last_name* and *nationality* with values Smith and US, respectively. Condition " $c_1 \triangleright type = \text{passport} \wedge c_1.last_name = \text{Smith} \wedge c_1.nationality = \text{US} \wedge c_2 \triangleright type = \text{credit_card} \wedge c_2.last_name = \text{Smith} \wedge c_2.cc_number$ " can be satisfied by two different credentials, the first one of *type* *passport*, containing attributes *last_name* and *nationality* with values Smith and US, respectively, and the second one of *type* *credit_card* containing attribute *last_name* with value Smith and reporting the credit card number (note that the expression is satisfied by the presence of attribute *cc_number*, regardless of its value).

Dealing with credentials requires distinguishing, in the language, between certified and uncertified properties, that is, properties that can be uttered by the requester but do not need to be certified. Requests for uncertified properties can be expressed by using simple uncertified conditions of the form ' A ' or ' $A \pi v$ ', where A is an attribute, π a standard predicate, and v a value or another attribute. As for credential conditions, also simple uncertified conditions can be combined by using the AND and OR boolean operators. For instance, " $last_name = \text{Smith} \wedge nationality = \text{US}$ " is satisfied if the requester declares (without any certification) that the *last_name* is Smith and the *nationality* is US.

3.2 Abstractions

An abstraction defines a shorthand for new concepts that can be expressed in terms of conditions on other concepts. Such abstractions simplify the specification of conditional expressions and provide a support for ontological reasoning. Formally, an abstraction is defined as follows.

Definition 3.4 (Abstraction): An abstraction is a rule of the form $\text{simple_cond} \leftarrow \text{cond}$, where *simple_cond* is simple credential or uncertified condition, and *cond* is a boolean expression of a credentials or uncertified conditions.

For instance, abstractions:

- $c \triangleright type = \text{id_document} \leftarrow c \triangleright type = \text{identity_card} \vee c \triangleright type = \text{passport} \vee c \triangleright type = \text{driver_license}$;
- $c \triangleright type = \text{emoney} \leftarrow c \triangleright type = \text{credit_card} \vee c \triangleright type = \text{debit_card} \vee c \triangleright type = \text{paypal}$

define *id_document* and *emoney* as two abstract credential types corresponding to any element in the sets of credentials {*identity_card*, *passport*, *driver_license*} and {*credit_card*, *debit_card*, *paypal*}, respectively. Hence, a request for an identifying document (credential of *type* *id_document*) can be satisfied by providing either an identity card, a passport, or a driver license. Abstractions can be exploited for defining and organizing concepts and taxonomies without the need for hierarchical data structures like in traditional ontologies.

3.3 Relation conditions and recursive conditions

One of the most interesting features offered by logic-based policy languages is represented by the support for recursive conditions. Recursion has a crucial role in the representation of restrictions on how authorities and, more in general, trusted parties delegate the ability to issue credentials. The delegation consists of a certification of the ability of another party to produce credentials on behalf of the delegator. In large scale distributed systems with a complex architecture, delegation increases flexibility and permits the inexpensive creation of credentials, particularly in an open environment. Such systems are characterized by application requirements calling for the specification of restrictions in delegation. The support for recursion in the policy language can be applied to the expression of conditions on data with a recursive structure.

Let us introduce the support for delegation. Let U be the set of all users that can take part in an access control process. Let $\rho \subseteq U \times U$ be a relation between elements in U . As an example to illustrate our ideas, let us focus on particular elements in U , namely, certification authorities, and let ρ be a relation that holds between two certification authorities u and v if and only if u has signed v 's public key on a certificate, delegating to v the authority to produce credentials, certified by v , that are to be considered as certified by u . In turn, v has the possibility to delegate her power to another certification authority, so that a chain of delegation is created, whose description must be maintained in some data structure

accessible by all the users that rely on the relevant certification authorities. Θ_ρ carries this information that exhaustively describes ρ , which we call the *context* of ρ , in the form of a sequence of credential-like entries corresponding to all the pairs that ρ induces in U : $\Theta_\rho = \{\theta = (\mathcal{M}_\theta, \mathcal{A}_\theta) : \underline{\mathcal{M}}_\theta = (rel), \theta \triangleright rel = rho, \underline{\mathcal{A}}_\theta = (user_1, user_2), (\theta.user_1, \theta.user_2) \in \rho\}$.

When ρ holds between u and v , that is, $(u, v) \in \rho$, we assume that there exists a $\theta \in \Theta_\rho$ such that $\theta \triangleright rel=rho$, $\theta.user_1 = u$, and $\theta.user_2 = v$. Conditions on data with a recursive structure like the one mentioned above can be requested in an access control policy. In our example, in which ρ is a relation of delegation between certification authorities, a requester trying to access a particular resource may be required by the server to show that the certification authority ca_r , signing her credentials has been delegated by a particular authority ca_s preferred by the server. The policy will then include the relevant relation condition, $\theta \triangleright rel=rho \wedge \theta.user_1 = ca_s \wedge \theta.user_2 = ca_r$, which, in general, can be rewritten according to the following abbreviation:

$$\theta.rho = \langle u, v \rangle \leftarrow \theta \triangleright rel=rho \wedge \theta.user_1 = u \wedge \theta.user_2 = v.$$

In this scenario, the need often arises to deal with the transitive closure of the delegation chain. Instead of setting conditions on the authority that directly delegated the one signing the requester's certificate, a server may be interested in ensuring that the root authority ca_{root} , the one at the very beginning of the delegation chain, is among her preferred ones. The requester can prove that her ca_r is in the relation ρ^* (i.e., the transitive closure of ρ) with ca_{root} either by showing that $(ca_{root}, ca_c) \in \rho$, or by providing a chain of context entries $\theta_1, \dots, \theta_n \in \Theta_\rho$, where ca_{root} is $user_1$ in θ_1 , ca_c is $user_2$ in θ_n , and for all $1 \leq i < n$, $\theta_i.user_2 = \theta_{i+1}.user_1$, which can be abbreviated in what we define as a *recursive condition*:

$$\theta.rho^* = \langle u, v \rangle \leftarrow \theta.rho = \langle u, v \rangle \vee (\theta.rho = \langle u, \theta'.user_1 \rangle \wedge \theta'.rho^* = \langle \theta'.user_2, v \rangle).$$

3.4 Dialog management

The ability to truly support an open world scenario implies enabling the servers to process requests for services coming from parties unknown a-priori. As noted, the way access control is enforced needs then to change. We can no longer assume that the server has available (either in its own state or released by the requester together with the request) all the information needed for evaluating access, and returns a definite decision. Rather, the server should be able to evaluate the policy with respect to unknown or partially known requesters, and communicate to them the conditions that need to be satisfied to access the service. XACML does not have these capabilities and returns indeterminate whenever the evaluation of the access control policy cannot be reduced to a definite state (permit or deny) and there are conditions whose truth value cannot be evaluated. Our goal is to depart from the indeterminate status allowing

the server to inform the requester of the conditions that it needs to satisfy instead of communicating it that there are conditions that cannot be evaluated.

An important issue is how the server should communicate its access control policy to the requester. For instance, suppose that an authorization imposes that attribute *nationality* should be equal to US. Should the server communicate such a condition to the requester? Or should it just inform the requester that it has to state the nationality? Clearly there is no unique response to whether one option is better than the other, and which one is to be preferred depends on the specific context and information involved. We can notice that communicating the complete policy (i.e., the fact that the policy will grant access if the nationality is US) favors the privacy of the requester. In fact, a requester can know, before releasing credentials or information to the server, whether the release will be sufficient to acquire access to the service. In particular, a client associated with a non-US user can avoid disclosing the nationality of the user. By contrast, communicating only part of the policy favors the privacy of the server. As a matter of fact, the access control policy, and the information on which it evaluates, can be considered sensitive too and as such needs to be protected. For instance, while the server might not mind disclosing the fact that access to a service is restricted to US citizens, it might not want to disclose other conditions (or values against which properties are evaluated) as they are considered sensitive. As an example, consider an authorization allowing access to a service to those users who work for an organization that does not appear in a Secret Black List (SBL) kept by the server. The corresponding subject expression is: $c \triangleright type = employment \wedge c.employer \notin SBL$. Communicating the complete policy to the requester (and allowing its evaluation by the requester) would imply releasing the subject expression, together with the state of black list SBL. Also, assuming the context of SBL is not released, the requester will know, in case it will not be granted access, that its employer is black listed. This is clearly an information the server does not wish to disclose; rather the server will want to maintain confidential the condition and simply state that the employment certificate is required. Between the two extremes of simply returning indeterminate (the XACML approach), on one side, and of completely disclosing the policy, on the other side, there are therefore other options offering different degrees of protection to the server policy and of information communicated to the requester. Each condition appearing in the policy can then be subject to a different disclosure policy, regulating the way the presence of such a condition should be communicated to the requester. We can distinguish five different disclosure policies, with each one potentially used independently in any condition appearing in an expression. In terms of our formal notation, we denote the disclosure policy by including the portion of a condition to not be disclosed in square brackets. For concreteness

TABLE 1
Disclosure policies and their effect on conditions

Disclosure policy	Condition in expression	Communication to the client
<i>none</i>	$[c \triangleright M \pi m]$ $[c.A \pi v]$	$[\]$ $[\]$
<i>credential</i>	$c \triangleright [M \pi m]$ $c.A \pi v$	$c \triangleright [\]$ $c.A \pi v$
<i>property</i>	$c \triangleright M [\pi m]$ $c.A \pi v$	$c \triangleright M [\]$ $c.A \pi v$
<i>predicate</i>	$c \triangleright M \pi [m]$ $c.A \pi v$	$c \triangleright M \pi [\]$ $c.A \pi v$
<i>condition</i>	$c \triangleright M \pi m$ $c.A \pi v$	$c \triangleright M \pi m$ $c.A \pi v$

of the discussion, we assume a condition $c \triangleright M \pi m$; the case of a condition on an attribute (i.e., of the form $c.A \pi v$ or $A \pi v$) is analogous. The following disclosure policies can be associated with the condition.

- *None*. Nothing can be disclosed about the condition. It corresponds to the XACML approach as only the information that the outcome of the policy is indeterminate is communicated, since there are conditions that cannot be evaluated. Formally, the condition will appear in the expression completely included in square brackets, that is, $[c \triangleright M \pi m]$.
- *Credential*. Only the information that there is a condition imposed on some metadata about a credential (or on some attributes of the credential) can be disclosed. The metadata (or attributes) on which the conditions are evaluated are not released. Formally, the condition will appear in the expression as $c \triangleright [M \pi m]$.
- *Property*. Only the information that a property (metadata or attributes of a credential, or uncertified statements) needs to be evaluated can be released; no information can be released on the control that will be enforced on the property. Formally, the condition will appear in the expression as $c \triangleright M [\pi m]$.
- *Predicate*. Only the information that a property (metadata or attributes of a credential, or uncertified statements) needs to be evaluated and the predicate with which it is evaluated can be released; no information can be released on the values against which the evaluation is performed. Formally, the condition will appear in the expression as $c \triangleright M \pi [m]$.
- *Condition*. The condition can be fully disclosed as it is. Formally, the condition will appear in the expression with no square brackets, signaling that no component is subject to disclosure restriction, that is, $c \triangleright M \pi m$.

Table 1 summarizes the different disclosure policies reporting the formal notation with which they appear in the expression and the consequent communication to the client in the dialog.

Note that the disclosure policies of the server, affect-

ing the information released to the requester about the conditions appearing in the policy, also impact the way the requester can satisfy the conditions. In particular, the credential policy implies that the requester will not know which information in the credential is needed and therefore will have to release the credential in its entirety (assuming that the credential to which the condition refers is known by other conditions in the policy, else the requester will have to disclose all its credentials). The property policy implies that the requester can selectively disclose the property in the credential (or utter it, in case of a condition on uncertified properties). The same for the predicate policy, where the requester however knows also against what predicate the property will be evaluated. Finally, in the case of the condition policy, the requester can provide either the property (but it can assess, before submitting, whether such a release will satisfy the condition) or a proof that the property is satisfied [4].

Example 3.1: Consider a policy stating that “a user can access a service if her nationality is Italian, her city of birth is Milan, and her year of birth is earlier than 1981”. Suppose that all attributes mentioned in the policy must be certified by an X.509 identity card or by a SAML passport both released by IT_Gov. The policy is formally stated as:

$$((c_1 \triangleright type = identity_card \wedge c_1 \triangleright method = X.509) \vee (c_1 \triangleright type = passport \wedge c_1 \triangleright method = SAML)) \wedge c_1 \triangleright issuer [= IT_Gov] \wedge c_1.nationality [= Italian] \wedge c_1.city_of_birth = Milan \wedge c_1.year_of_birth < [1981])$$

Here, the square brackets representing the disclosure policies implicitly state that: i) conditions on metadata type and method, and attribute city_of_birth can be eventually disclosed as they are; conditions on metadata issuer and attribute nationality need to be protected by hiding the control that will be enforced on them; and iii) condition on attribute year_of_birth needs to be protected by hiding the value against which the evaluation will be performed. If the above policy applies to a request submitted by a requester for which the server has no information, the following conditions are communicated to the requester.

$$((c_1 \triangleright type = identity_card \wedge c_1 \triangleright method = X.509) \vee (c_1 \triangleright type = passport \wedge c_1 \triangleright method = SAML)) \wedge c_1 \triangleright issuer [\] \wedge c_1.nationality [\] \wedge c_1.city_of_birth = Milan \wedge c_1.year_of_birth < [\]$$

The requester can satisfy such conditions by releasing either an identity card or a passport containing the requested attributes.

4 DEPLOYMENT IN XACML

We illustrate how the concepts introduced and formalized in the previous section can be concretely deployed within XACML. Our solution consists in defining new components or in using, in a different way, existing components of the language. An important characteristic of our deployment is that it has a limited impact on the original XACML specification.

4.1 Basic XACML concepts

XACML relies on a model that provides a formal representation of access control policies and on mechanisms for their evaluation. A XACML policy contains one Policy or PolicySet root element, which is a container for other Policy or PolicySet elements. Element Policy consists of a Target, a set of Rule, an optional set of Obligation, and a rule combining algorithm. A Target element includes simple conditions on Subject, Resource, Action, and Environment. If a request satisfies the conditions specified in the Target, the corresponding policy applies to the request. A Rule corresponds to a positive (permit) or a negative (deny) authorization, depending on its effect, and may include an element Target, and an element Condition specifying further restrictions on subject, resource, and action. Each condition can be defined through element Apply with attribute *FunctionID* denoting the XACML predicate (e.g., string-equal, integer-less-than) and with appropriate sub-elements denoting both the attribute against which the condition is evaluated and the comparison value. The rule's effect is then returned whenever the rule evaluates to true. The Obligation element specifies an action that has to be performed in conjunction with the enforcement of an authorization decision. Each element Policy has attribute *RuleCombiningAlgID* specifying how to combine the decisions of different rules to obtain a final decision of the policy evaluation (e.g., deny overrides, permit overrides, first applicable, only one applicable). According to the selected combining algorithm, the authorization decision can be permit, deny, not applicable (i.e., no applicable policies or rules can be found), or indeterminate (i.e., some information is missing for the completion of the evaluation process).

4.2 Support for credentials

In XACML, properties appearing in conditions are typically uncertified. To represent credentials, and therefore relevant properties and conditions, with minimal impact on the XACML specification, we consider splitting the conditions on metadata from the conditions on the attributes. Attributes in the credentials can be treated just like any other property. The only change needed for their treatment is that they must be associated with a certificate. To this purpose, we exploit the *Issuer* attribute that is included in the SubjectAttributeDesignator element of XACML. In particular, an occurrence of attribute *c.A* in the subject expression will translate into an element SubjectAttributeDesignator, where attribute *Attributeld* is equal to *A*, and attribute *Issuer* refers to *c*. Every credential symbol appearing in a subject expression translates to a value of the *Issuer* attribute, and different credential symbols translate to different values. Hence, attributes referring to the same credential (e.g., $c.A_i$ and $c.A_j$) will have the same value *c* in their attribute *Issuer*, while attributes referring to different credentials (e.g., $c_i.A_i$ and $c_j.A_j$) will not. This binding of

```
<certifications>
  <certification id="IT_IC">
    <group>
      <type Disclosure="condition">identity_card</type>
      <issuer Disclosure="property">IT_Gov</issuer>
      <method Disclosure="condition">X.509</method>
    </group>
    <group>
      <type Disclosure="condition">passport</type>
      <issuer Disclosure="property">IT_Gov</issuer>
      <method Disclosure="condition">SAML</method>
    </group>
  </certification>
  <certification id="VISA_CC">
    <group>
      <type Disclosure="condition">credit_card</type>
      <issuer Disclosure="condition">VISA</issuer>
      <method Disclosure="condition">SAML</method>
    </group>
  </certification>
</certifications>
```

Fig. 1. XML representation of conditions on credentials metadata

attributes to credentials implicitly guarantees binding of attributes that must be certified by the same credential.

To represent credentials, and relevant metadata, we introduce a new XML schema, whose document instances correspond to the specification of credential conditions on metadata. The schema defines certifications as the root element containing one or more certification elements. Each certification corresponds to the definition of a credential condition on metadata. A certification element has an attribute *id* whose value is the identifier by which the credential condition should be referred to. In other words, it defines the values to be used in the *Issuer* attribute of a SubjectAttributeDesignator to state that it must be certified by a credential satisfying the corresponding conditions on metadata. A certification element includes one or more group elements, whose content depends on the considered metadata. Conditions expressed by elements within a group are considered in conjunction, while different elements group are considered in disjunction.

Figure 1 illustrates an example of metadata conditions. Element certification with attribute *id* equals to IT_IC represents either a credential of type identity_card issued by IT_Gov with the X.509 proof method, or a credential of type passport issued by IT_Gov with the SAML proof method. Element certification with attribute *id* equals to VISA_CC represents a credential of type credit_card issued by VISA with the SAML proof method.

Figure 2 presents the XACML-based representation of the policy in Example 3.1 stating that “a user can access a service if her *nationality* is Italian, *city of birth* is Milan, and *year of birth* is less than 1981”, and that all attributes/conditions must be certified via an X.509 identity card or via a SAML passport both released by IT_Gov. Since all attributes refer to the same element certification (i.e., the certification element whose attribute *id* is equal to IT_IC), the conditions defined on such attributes have to be satisfied by attributes included

```

<Rule RuleId="ExampleRule" Effect="Permit">
  <Target><!-- all target fields --></Target>
  <Condition
    FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply Disclosure="property"
      FunctionId="urn:oasis:names:tc:xacml:2.0:function:string-equal">
      <SubjectAttributeDesignator
        DataType="http://www.w3.org/2001/XMLSchema#string"
        Issuer="urn:ext:cred-reference:IT_IC"
        AttributeId="urn:oasis:names:tc:xacml:2.0:attribute:nationality"/>
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
        Italian
      </AttributeValue>
    </Apply>
    <Apply Disclosure="condition"
      FunctionId="urn:oasis:names:tc:xacml:2.0:function:string-equal">
      <SubjectAttributeDesignator
        DataType="http://www.w3.org/2001/XMLSchema#string"
        Issuer="urn:ext:cred-reference:IT_IC"
        AttributeId="urn:oasis:names:tc:xacml:2.0:attribute:city-birth"/>
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
        Milan
      </AttributeValue>
    </Apply>
    <Apply Disclosure="predicate"
      FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-less-than">
      <SubjectAttributeDesignator
        DataType="http://www.w3.org/2001/XMLSchema#integer"
        Issuer="urn:ext:cred-reference:IT_IC"
        AttributeId="urn:oasis:names:tc:xacml:2.0:attribute:year-birth"/>
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#integer">
        1981
      </AttributeValue>
    </Apply>
  </Condition>
</Rule>

```

Fig. 2. An example of XACML policy

in a single credential that may be *either* an identity card *or* a passport.

4.3 Support for abstractions

In Section 3, we introduced the concept of abstraction ($sc \leftarrow co$) as the means to perform ontological reasoning, and to specify compact and abstract policies. To manage abstraction specification in XACML, we prescribe the integration of XACML with XQuery.

XQuery is a language developed by W3C for querying XML data. XQuery 1.0 [5] reached the status of recommendation in 2007. XQuery has been designed taking into account the experience of the database community and presents strong similarities with SQL: queries, for example, are declarative and the execution plan is produced by a query optimizer. XQuery follows the functional programming paradigm. Even if the specification is relatively recent, there are already several XQuery engines and many experts envision a significant role for the language in the context of XML and Web technology. Since XACML is a declarative XML language for controlling access, it is quite natural to consider its integration with XQuery. In the following, we show that the use of XQuery provides an interesting solution to the management of abstractions. Recursion is also supported by XQuery (Section 4.4).

Abstractions can be defined via XQuery functions and

```

declare function local:expand($x as xs:string*) as xs:string* {
  let $m := $doc//abstractions/abstraction

  for $i in $x
  return
    if ( 0 = count($m[@id=$i]/is/item) ) then $i
    else $m[@id = $i]/is/item/text()
};

```

Fig. 3. XQuery function *expand*

```

<abstractions>
  <abstraction id="id_document">
    <is>
      <item>identity_card</item>
      <item>driver_license</item>
      <item>passport</item>
    </is>
  </abstraction>
  <abstraction id="emoney">
    <is>
      <item>credit_card</item>
      <item>debit_card</item>
      <item>paypal</item>
    </is>
  </abstraction>
</abstractions>

```

Fig. 4. XML representation of two abstractions

embedded in XACML conditions. Our solution consists in the definition of an XQuery function called *expand* (see Figure 3), which takes in input abstraction head *sc* and produces in output abstraction tail *co*.

To specify abstractions in the definition of a condition, let us define an XML schema that models the abstractions to be used. This schema includes a root element called *abstractions* containing a number of abstraction elements, each with an *id* (head *sc*), and a set of equivalences in the *is* element (tail *co*). Figure 4 illustrates an example of two abstractions. The first abstraction shows the equivalence between *id_document* and $\{\text{identity_card, driver_license, passport}\}$, while the second one shows the equivalence between *emoney* and $\{\text{credit_card, debit_card, paypal}\}$. Based on the proposed schema and on function *expand*, an XQuery invocation that refers to an abstraction via its *id* can be defined both in a XACML condition, and in an XML-based metadata condition. At evaluation time, the server applies function *expand* to the abstraction in the policy and compares the expanded abstraction with the data in the context. If the data are not sufficient, the abstraction must be presented to the requester. The requester however does not have to know the meaning of the abstraction, and therefore may not be able to release the proper credential for satisfying it. As a consequence, the requester will receive a request including all the conditions in the tail of the abstraction. Figure 5 illustrates an example of condition stating that the value of attribute *last_name* has to be equal to Smith and that *last_name* must be certified through a certificate of type *id_document* corresponding to the first abstraction in Figure 4. At evaluation time, the server will apply function *expand* to the condition on metadata type and will search for the *last_name* in an

```

<Condition Disclosure="property"
  FunctionId="urn:oasis:names:tc:xacml:2.0:function:string-equal">
  <SubjectAttributeDesignator
    DataType="http://www.w3.org/2001/XMLSchema#string"
    Issuer="urn:ext:cred-reference:IT_ABBR"
    AttributeId="urn:oasis:names:tc:xacml:2.0:attribute:last_name"/>
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">
    Smith
  </AttributeValue>
</Condition>

```

(a)

```

<certifications>
  <certification id="IT_ABBR">
    <group>
      <type Disclosure="condition">
        local:expand('id_document')
      </type>
    </group>
  </certification>
</certifications>

```

(b)

Fig. 5. An example of XACML condition (a) and of abstraction-based metadata condition (b)

identity_card, a driver_license, or a passport of the requester. If such information is not available, the requester will receive a request for attribute *last_name* certified by an identity_card, a driver_license, or a passport.

4.4 Support for recursion

As we discussed in Section 3.3, recursion is a useful construct in the definition of access control policies, specifically to manage situations involving chains of credentials and recursive relations. Current XACML implementations impose a flat context as the only source of data. Data used in policy evaluation must be materialized in the context and complex reasoning is not supported.

Let us first consider the scenario where a verification of credential chains is requested. The server may ask a client to provide a credential certified directly by a given authority *CA*, or alternatively to provide a credential certified by an authority that belongs to a chain referring to *CA*. Upon receiving the credentials, the server should be able to check, by means of a trusted party, whether they have been released directly, or indirectly, through a chain of credentials. To illustrate in a simple way how a recursive reasoning can be supported in XACML, we consider a scenario where a server has to evaluate conditions requiring a recursive reasoning on data. An example could also be realized for detailing restrictions on the chain of delegations. However, it would be only more complex than the example that we will describe in the following, without providing any additional insight.

Suppose therefore that a policy states that all supervisors of a doctor *d* can read patients' records of *d*. We rely on the assumption that the XACML context contains the complete description of the relation between each doctor and her direct supervisor. In this case, XACML should

be able to evaluate conditions that need recursion over the data, since the supervisor of a supervisor of *d*, is also a supervisor of *d*.

We propose to manage recursion in XACML using the services of an XQuery engine in a similar way to what is done for abstractions. Recursive conditions are realized by defining recursive XQuery functions, which are then embedded and referenced in the policies, without changes to the XACML language. XQuery functions are used to define complex and recursive concepts to be used in the definition of the policy conditions. These functions take in input the XACML context, and produce new information to be used in policy evaluation. As a consequence, XQuery offers recursive reasoning and allows for the just in time creation of additional attributes to be used in the evaluation of the XACML policies.

A working example presents how XQuery can be used to manage a recursive policy stating that "all supervisors of doctor *d* can read patients' records of *d*".

First, consider the XACML context in Figure 6, which contains identity information of doctors, including the *supervisor* relation. For instance, doctor George Williams, specialized in surgery, is a supervisor of doctor Charles White, specialized in pediatric surgery; therefore, doctor White may authorize doctor Williams to access his patients data.

Figure 7 defines an XQuery function to reason about the context and to retrieve information about the chain of supervisor relations. Variable `$doc` represents the whole XACML context. As said before, the supervisor relation is recursive. Then, if we authorize access to the data of the patients of *d* to *d*'s supervisors, we also authorize the supervisor of *d*'s supervisors. Note that, in the XQuery processing approach, no extension to the context is provided; the information needed for policy evaluation is generated just in time, exploited in the policy evaluation process, and then discarded.

Figure 8 provides an example of a XACML condition where doctors who are direct or indirect supervisors of the doctor of a patient are granted access to the patient's data. The XQuery function in the XACML condition is simply invoked as a part of the XPath URI. In fact, at evaluation time, the `AttributeSelector` field is generated by applying function `getSupervisor` (see Figure 7) that takes as input the element `doctor` retrieved via URI:

```

//doctor[@id=//patient[@id=
urn:oasis:names:tc:xacml:2.0:attribute:patient-id]/doctorid]

```

identifying the doctor of the patient for which access to data is requested. Function `getSupervisor` then returns all doctors that are direct or indirect supervisors of such a doctor, and from these doctors attribute *id* (i.e., `/@id`) is selected. Here, we assume that `//patient` is the portion of the context that contains data of the patient.

XQuery constructs can be integrated within XACML with a minimal impact on the language. Our approach requires a single extension, that is, the definition of rules presenting references to XQuery functions.

```

<context>
  <doctor id="1">
    <first_name>George</first_name>
    <last_name>Williams</last_name>
    <specialized>Surgery</specialized>
    <sex>M</sex>
    <supervisor/>
  </doctor>
  <doctor id="2">
    <first_name>Charles</first_name>
    <last_name>White</last_name>
    <specialized>Pediatric Surgery</specialized>
    <sex>M</sex>
    <supervisor>
      <doctorid>1</doctorid>
    </supervisor>
  </doctor>
  <doctor id="3">
    <first_name>Mary</first_name>
    <last_name>Wilson</last_name>
    <specialized>Pediatric Allergy</specialized>
    <sex>F</sex>
    <supervisor>
      <doctorid>1</doctorid>
    </supervisor>
  </doctor>
  <doctor id="4">
    <first_name>Paul</first_name>
    <last_name>Brown</last_name>
    <specialized>Kidney Ailments</specialized>
    <sex>M</sex>
    <supervisor>
      <doctorid>3</doctorid>
    </supervisor>
  </doctor>
  <doctor id="5">
    <first_name>Dorothy</first_name>
    <last_name>Jackson</last_name>
    <specialized>Fitness and Nutrition</specialized>
    <sex>F</sex>
    <supervisor/>
  </doctor>
</context>

```

Fig. 6. An example of XACML context

```

declare function local:getSupervisor($Doctors as element(*) as element)* {
  let $c := $doc//doctor

  for $d in $Doctors
  return
    if ( 0 = count($d/supervisor/doctorid) ) then $d
    else $d | local:getSupervisor($c[@id = $d/supervisor/doctorid])
};

```

Fig. 7. XQuery function *getSupervisor*

```

<Condition
  FunctionId="urn:oasis:names:tc:xacml:2.0:function:string-equal">
  <SubjectAttributeDesignator
    DataType="http://www.w3.org/2001/XMLSchema#string"
    AttributeId="urn:oasis:names:tc:xacml:2.0:attribute:doctor-id"/>
  <AttributeSelector RequestContextPath=
    "local:getSupervisor(//doctor[@id=
    //patient[@id=urn:oasis:names:tc:xacml:2.0:attribute:patient-id]
    /doctorid])/@id"
    DataType="http://www.w3.org/2001/XMLSchema#string"/>
</Condition>

```

Fig. 8. An example of XACML condition invoking the XQuery function in Figure 7

We conclude this section with a note on performance. One can be afraid that the integration of a powerful query language in XACML could cause excessive computational load on the system in the access control

evaluation. Still, the Web service scenario we are considering typically offers large computational resources and the gain in terms of flexibility of the access control language clearly overcomes possible worries about the load incurred. We also observe that the explicit use of XQuery recursion exploits all the benefits that derive from its robust implementation in the language. XQuery engines are implemented taking into account all the work done in the last 20 years in the database community for the management of recursive queries, which has produced support for recursive SQL queries in most recent relational DBMSs. Such sophisticated strategies are far from trivial, and in the approach we present they are immediately available in the evaluation of expressive security policies. In general, the proposal is consistent with the long term evolution of access policies, for which a stricter integration with declarative query languages is more and more often considered also in the context of databases. Compared to the database scenario, here we have an even stronger motivation for the use of declarative query languages, due to the openness of the environment and the need for supporting more flexible identification services.

4.5 Support for dialog

Support for dialog requires both to introduce a change in the XACML language for representing the disclosure policies associated with conditions and to define how the different disclosure policies are effectively enforced (note that changes requested to the XACML architecture will be discussed in Section 5). For each condition appearing in a XACML policy, the corresponding disclosure policy is represented through a new attribute *Disclosure* added to elements *Condition* and *Apply* of XACML, and to each sub-element of *group*. The admissible values for such an attribute are the disclosure policies presented in Section 3.4. For instance, certification VISA_CC in Figure 1 includes three conditions on *type*, *issuer*, and *method*, respectively, that are associated with disclosure policy condition, as represented by the value of their attributes *Disclosure*. Analogously, in the XACML policy in Figure 2, we have a conjunction of three conditions on attributes *nationality*, *city-birth*, and *year-birth*, respectively, that, according to the value of attribute *Disclosure* of the corresponding elements *Apply*, are associated with disclosure policies *property*, *condition*, and *predicate*, respectively.

A disclosure policy that specifies what information in a condition cannot be released is then enforced by hiding such information. To this purpose, a special keyword *undisclosed* will be used in the XACML policy to indicate information hidden to comply with the selected disclosure policy. Keyword *undisclosed* is equivalent to the empty square brackets in our formal model. For instance, consider again the XACML policy in Figure 2. If the information needed for policy evaluation is not available at the server side, the server communicates to

TABLE 2
From formal concepts to XACML

Basic Constructs	XACML
Declaration conditions	No change required
Certification $c \triangleright M \pi m$	$\langle \text{certification} \rangle$... $\langle / \text{certification} \rangle$
Credential conditions $c \triangleright M \pi m \wedge c.A \pi v$	Attribute <i>Issuer</i> in element $\langle \text{SubjectAttributeDesignator} \rangle$
Abstractions	XQuery functions
Recursive statements	XQuery functions
Dialog management	Attribute <i>Disclosure</i> in any of: - element $\langle \text{Condition} \rangle$ in XACML - element $\langle \text{Apply} \rangle$ in XACML - sub-elements of $\langle \text{group} \rangle$

the requester the policy, where each condition is transformed according to the corresponding disclosure policy. In our example, since the disclosure policy is property, the first condition on attribute *nationality* is transformed by hiding, that is, by setting to undisclosed the value of attribute *FunctionId* and the value Italian. The remaining two conditions are transformed in an analogous way by applying the corresponding disclosure policies.

It is important to note that whenever a policy includes a condition that involves an abstraction, the abstraction is first expanded by replacing its occurrences in the condition with the corresponding expansion. Then, the disclosure policy associated with the original condition is applied to each component of the expansion. For instance, consider the XACML condition in Figure 5(a) that defines a condition on attribute *last_name* that should be certified via abstraction *id_document* represented in Figure 5(b). Abstraction *id_document* is expanded to *identity_card*, *driver_license*, and *passport*. No change is required on the expansion since the specified disclosure policy is condition; for disclosure policies different from condition, the transformation must be applied to the components resulting by the expansion of *id_document*.

Table 2 summarizes the mapping between the basic constructs of our formal model and the changes introduced in XACML.

5 EXTENDED XACML ARCHITECTURE

To support the new functionalities of the XACML language introduced in the previous section, we present a possible extension of the XACML architecture. In the following, we first present the standard XACML architecture, and then describe the new components and data flows of the extended architecture.

5.1 Standard XACML architecture

XACML defines both a modular and distributed architecture for the evaluation of policies, and a communication protocol for message interchange. The architecture

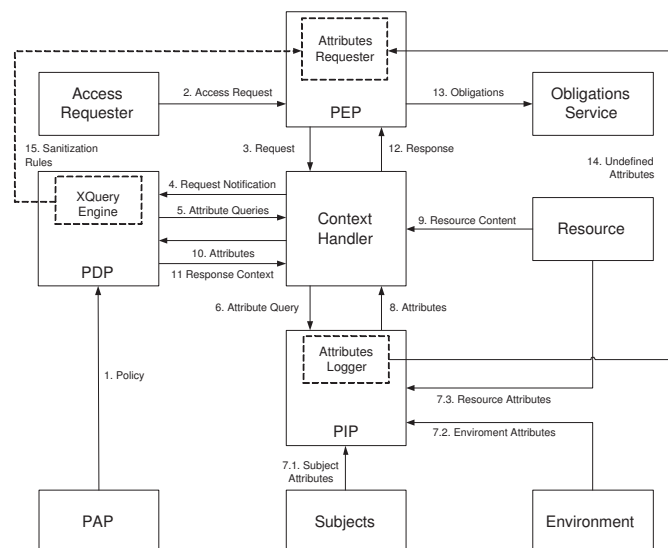


Fig. 9. Extended XACML architecture

includes different functional components that interact to take an access control decision. Figure 9 illustrates the standard XACML architecture with solid lines while our extensions are represented through dashed lines. We now focus our discussion on the standard XACML architecture.

Each access request is received by the *Policy Enforcement Point* (PEP), that is responsible for the enforcement of each access decision. The *Policy Decision Point* (PDP) is the component that produces the access decision for each access request by retrieving the applicable policies from the *Policy Administration Point* (PAP), that provides functionalities to administer access control policies. The PDP must receive all information relevant for the decision process. To this aim, the *Context Handler* manages the data released directly by the requester, and provides the interface to access additional information. In addition to what is included in the access request, in fact, the PDP may also need to get information via the *Policy Information Point* (PIP), that is the component acting as a source of attribute values. PIP retrieves the attribute values related to the subject, the resource, or the environment from the system information storage (e.g., DBMS).

Data flow in the standard XACML model can be summarized as follows.

- The requester sends an access request to the PEP module (*step 2*), which has to enforce the access decision that will be taken by the PDP. The requester has to release all relevant information for policy evaluation.
- The PEP module sends the access request to the Context Handler (*step 3*) that translates the original request into a canonical format, called XACML request context, and sends it to the PDP (*step 4*).
- The PDP identifies the applicable policies by means of the PAP module (*step 1*), and retrieves the required attributes and, possibly, the resource by

means of the Context Handler. The context handler also relies on the PIP module (*step 5-10*) to access attribute values about the subject, resource, action, and environment. To this purpose, the PIP interacts with the Subject, Resource, and Environment modules. The Environment module provides a set of attributes that are relevant to take an access decision and are independent from a particular subject, resource, or action.

- The PDP evaluates the policies and returns the XACML response context, together with an optional set of obligations, to the Context Handler (*step 11*). The Context Handler translates the XACML response context to the native format of the PEP and returns it to the PEP (*step 12*). If some information is missing, the PDP cannot take a decision, and returns an error (*Indeterminate* response).
- The PEP fulfills the obligations (*step 13*) and, if permitted, it gives access to the requester. Otherwise, access is denied.

5.2 An extended XACML architecture to support language improvements

We describe the extensions we propose to the standard XACML architecture, which, as already mentioned above, are represented through dashed lines in Figure 9.

The integration of credential definition and evaluation in XACML (see Section 4.2) requires changes in the PDP component. The PDP is extended to support the evaluation of conditions that express restrictions on the certification mechanisms. To this aim, the PDP has to:

- understand metadata conditions defined in the certification document,
- retrieve data about certifications stored in the context or PIP extensions,
- evaluate conditions based on metadata restrictions (i.e., $c \triangleright M \pi m$).

The definition of an ad-hoc XACML profile for supporting the specification of credential conditions (see Figure 1) minimizes the changes needed to the core XACML language, but it requires the addition of an ad-hoc parsing process in the PDP. A possible solution to limit architectural changes is to define credential conditions by means of traditional XACML conditions. This solution may however result in inconsistencies due to the fact that XACML conditions permit the definition of complex restrictions also involving negations and negative authorizations. To limit, on one side, changes required to the PDP evaluation mechanism, and, on the other side, the possibility of wrong policy definition, we can adopt a hybrid solution where:

- 1) restrictions on the credential metadata are defined using the ad-hoc XACML profile that we have designed,
- 2) an XSLT style-sheet is defined to translate the ad-hoc syntax into the traditional XACML condition syntax.

This solution allows for the re-use of most part of the XACML evaluation mechanism, still limiting the complexity of the policy definition. In summary, the PDP has to be enriched with the functionality to transform, via XSLT, the certification document into a schema based on XACML conditions.

The integration of the XQuery-based approach for defining and evaluating complex and recursive conditions, discussed in Sections 4.3 and 4.4, calls for the development of an XQuery engine in the PDP component. The PDP enhanced with the XQuery engine manages the integration and execution of XQuery functions specified in XACML conditions. It actively evaluates the XQuery-based conditions by executing the XQuery functions contained in an XPath URI used to express each attribute. To this aim, the XQuery engine needs to access the Context Handler to retrieve all relevant attributes for policy evaluation. Also, the PDP retrieves the XQuery function definitions stored either in the XACML policies or in an ad-hoc profile, and uses them to expand and evaluate the policies. The XQuery extension is also suitable for the definition and evaluation of conditions involving XQuery-based abstractions (Section 4.3). The XQuery engine expands XQuery abstractions when they are used in XACML policies. Like with recursive XQuery functions, when the PDP needs to reason on abstractions, it extracts the definition of the abstractions stored in the XACML policies or in a XACML profile, and processes them in the policy under evaluation.

In general, the integration of credentials, recursive conditions, and abstractions in XACML only requires the implementation of an enhanced PDP, without modifying the communication flow of the standard XACML architecture, whereas the enhancement needed for dialog management (Section 4.5) calls for extensions to both XACML standard components and the communication flow.

An enhanced PIP has to be designed to identify and store all the attributes that are not available at evaluation time and need to be requested to the client. To manage such a case, resulting in an indeterminate evaluation returned by the PDP to the PEP via the Context Handler, the standard PEP must be extended with a direct communication channel to the PIP, to retrieve the list of missing attributes (*step 14*). In this way, the PEP collects all the information that must be required to the requester to complete the evaluation process. Before communicating with the requester, the PEP needs to retrieve from the PDP the conditions associated with each missing attribute, and the meta-information regarding the disclosure policy to be applied to such conditions (Section 4.5). Intuitively, the communication flow must be enriched to allow for direct communication between PDP and PEP (*step 15*). As discussed in Section 4.3, the dialog also supports XQuery-based abstractions: the requester will receive a request for attributes and credentials after the relevant abstractions have been expanded. After collecting all conditions for which some attributes are missing,

and applying the disclosure policy, a response that calls for additional information is returned to the requester. To support dialog management on the client-side, the client application must be extended to understand and fulfill such requests.

6 RELATED WORK

Several access control models presented in the literature are based on logic expressions, and prescribe access decisions on the basis of some properties of the requesting party, which can be proven by presenting one or more certificates (e.g., [6]–[10]). All these solutions are typically based on logic-based access control languages, which are powerful, highly expressive, and permit to specify recursive conditions and complex relations between parties in a simple yet effective way. While powerful, logic-based solutions may not always be applicable in real world scenarios like the one discussed in this paper, where simplicity and easiness are a must. In this work we then depart from logic-based paradigm to provide a solution that balances usability and expressiveness.

Another important research line focuses on providing solutions and protocols for trust negotiation [11], [12]. Several works (e.g., [10], [11]) investigated trust issues and strategies that a party can apply for selecting credentials that have to be submitted to the opponent party during a negotiation. These works view a negotiation process as a bidirectional exchange of request-response messages, where trust can be established by gradually disclosing data. In the context of open web services, however, trust negotiation approaches are not suitable for a wide range of use cases that request low complexity, low overhead, and high performance.

The research community has then focused on the definition of access control languages for an open world scenario, which provide flexibility and interoperability [1], [13]. eXtensible Access Control Markup Language (XACML) [1] is an XML-based language for expressing and interchanging access control policies. XACML has gained popularity, and has been extensively adopted in real world implementations dealing with open scenarios. However, XACML still has some important drawbacks related to the support of credential-based conditions, the definition of complex conditions (e.g., based on recursion), and the management of the dialog between the involved parties. In this context, the efforts in [14]–[16] provided XACML extensions to support trust negotiation based on its language and architecture. Chadwick et al. [17] described a mechanism to integrate XACML with the support for delegation of authority with chains of credentials issued from one user to another. The solution proposed is based on a Credential Validation Service (CVS) that works with the PDP in the evaluation of the policies.

The Web Services profile of XACML (WS-XACML) [18] aims at defining the aspects related

to authorization, access control, and privacy policy in a Web Service environment. This profile is meant to be used in combination with the WS-Policy proposal. WS-XACML is supposed to wrap XACML policies in a standardized format that can be put in a WS-Policy framework in alternative to policies from various other domains. Another relevant WS-* effort is the Web Service Security (WSS) specification [19], which prescribes a standard set of SOAP extensions to build secure Web Services ensuring message integrity and confidentiality. The authorization language we propose is expressive enough to include references to the security domain currently dealt with by the WSS proposal, which our XACML extension is not meant to counter, but to integrate. The Enterprise Privacy Authorization Language (EPAL) [20] is a formal language for writing enterprise privacy policies to govern data handling practices in IT systems. EPAL presents some relevant differences from XACML since it is specifically enterprise-oriented and proprietary [21].

7 CONCLUSIONS

The proposal presented in the paper is the result of our efforts within PrimeLife, a large EU-funded project with several industrial and academic partners, which is studying and developing novel technological solutions for privacy management. The specific focus of the project is toward the definition of a novel policy language and architecture able to support the privacy requirements of individuals interested in exploiting the services of the Web, keeping control on their data. In this scenario, XACML has been identified as a promising solution thanks to its technical features and the widespread support. However, its application in complex advanced scenarios, like the one considered in our project, requires overcoming the limitations discussed, and addressed, in this paper. The project is committed to building a robust prototype of a system supporting the management of privacy policies. An early implementation effort has already demonstrated the feasibility of the strategy presented in the paper and the absence of obstacles to its deployment.

ACKNOWLEDGMENT

This work was supported in part by the EU within the 7FP project “PrimeLife” under grant agreement 216483.

REFERENCES

- [1] T. Moses, *eXtensible Access Control Markup Language (XACML) Version 2.0*, OASIS, 2005.
- [2] A. Anderson and H. Lockhart, *SAML 2.0 profile of XACML*, OASIS, September 2004.
- [3] C. Ardagna, J. Camenisch, M. Kohlweiss, R. Leenes, G. Neven, B. Priem, P. Samarati, D. Sommer, and M. Verdicchio, “Exploiting cryptography for privacy-enhanced access control: A result of the prime project,” *Journal of Computer Security (JCS)*, vol. 18, no. 1, pp. 123–160, 2010.

- [4] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *Proc. of EUROCRYPT 2001*, Innsbruck, Austria, May 2001.
- [5] S. Boag et al., *XQuery 1.0: An XML Query Language*, World Wide Web Consortium (W3C), 2007.
- [6] P. Bonatti and P. Samarati, "A unified framework for regulating access and information release on the Web," *Journal of Computer Security*, vol. 10, no. 3, pp. 241–272, 2002.
- [7] K. Irwin and T. Yu, "Preventing attribute information leakage in automated trust negotiation," in *Proc. of ACM CCS 2005*, Alexandria, VA, USA, November 2005.
- [8] S. Jajodia, P. Samarati, M. Sapino, and V. Subrahmanian, "Flexible support for multiple access control policies," *ACM Transactions on Database Systems*, vol. 26, no. 2, pp. 214–260, June 2001.
- [9] M. Winslett, N. Ching, V. Jones, and I. Slepchin, "Assuring security and privacy for digital library transactions on the Web: Client and server security policies," in *Proc. of ADL 1997*, Washington, DC, USA, May 1997.
- [10] T. Yu, M. Winslett, and K. Seamons, "Supporting structured credentials and sensitive policies through interoperable strategies for automated trust," *ACM TISSEC*, vol. 6, no. 1, pp. 1–42, February 2003.
- [11] P. Bonatti and D. Olmedilla, "Driving and monitoring provisional trust negotiation with metapolicies," in *Proc of POLICY 2005*, Stockholm, Sweden, June 2005.
- [12] K. Seamons, M. Winslett, and T. Yu, "Limiting the disclosure of access control policies during automated trust negotiation," in *Proc. of NDSS 2001*, San Diego, CA, USA, April 2001.
- [13] C. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati, "A privacy-aware access control system," *Journal of Computer Security*, vol. 16, no. 4, pp. 369–392, 2008.
- [14] V. Cheng, P. Hung, and D. Chiu, "Enabling web services policy negotiation with privacy preserved using XACML," in *Proc. of HICSS 2007*, Hawaii, USA, January 2007.
- [15] D. Haidar, N. Cuppens, F. Cuppens, and H. Debar, "XeNA: an access negotiation framework using XACML," *Annals of telecommunications*, vol. 64, no. 1–2, January 2009.
- [16] U. Mbanaso, G. Cooper, D. Chadwick, and S. Proctor, "Privacy preserving trust authorization framework using XACML," in *Proc. of WOWMOM 2006*, Niagara-Falls, USA, June 2006.
- [17] D. Chadwick, S. Otenko, and T. Nguyen, "Adding support to XACML for dynamic delegation of authority in multiple domains," in *Proc. of CMS 2006*, Heraklion, Crete, Greece, October 2006.
- [18] O. X. T. Committee, *Web Services Profile of XACML (WS-XACML) Version 1.0*, OASIS, 2006.
- [19] O. W. T. Committee, *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*, OASIS, 2006.
- [20] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter, *Enterprise Privacy Authorization Language (EPAL)*, Research Report RZ 3485, IBM Research, March 2003.
- [21] A. Anderson, *A comparison of two privacy policy languages: EPAL and XACML*, Sun Microsystems, 2005.
- [22] A. Singhal, T. Winograd, and K. Scarfone, *Guide to Secure Web Services. Recommendations of the National Institute of Standards and Technology*, National Institute of Standards and Technology (NIST), Special Publication 800-95, 2007.
- [23] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati, "Controlling access to XML documents," *IEEE Internet Computing*, vol. 5, no. 6, November/December 2001.
- [24] —, "A fine-grained access control system for XML documents," *ACM TISSEC*, vol. 5, no. 2, May 2002.
- [25] C. Farkas and M. Huhns, "Securing enterprise applications: Service-oriented security (SOS)," in *Proc. of IEEE CEC/EEE 2008*, Washington, USA, July 2008.
- [26] C. Gutierrez, E. Fernandez-Medina, and M. Piattini, "A survey of web services security," in *Proc. of ICCSA*, Assisi, Italy, May 2004.
- [27] P. Belsis, S. Gritzalis, C. Skourlas, and V. Tsoukalas, "Design and implementation of distributed access control infrastructures for federations of autonomous domains," in *Proc. of TrustBus 2007*, Regensburg, Germany, September 2007.
- [28] M. Murata, A. Tozawa, M. Kudo, and S. Hada, "XML access control using static analysis," *ACM TISSEC*, vol. 9, no. 3, pp. 292–324, August 2006.



Claudio A. Ardagna is an assistant professor at the Department of Information Technology, Università degli Studi di Milano, Italy. His research interests are in the area of information security, privacy, access control, mobile networks, and open source. He is the recipient of the ERCIM STM WG 2009 Award for the Best Ph.D. Thesis on Security and Trust Management.
<http://www.dti.unimi.it/ardagna>



Sabrina De Capitani di Vimercati is a professor at the Department of Information Technology, Università degli Studi di Milano, Italy. Her research interests are in the area of information security, databases, and information systems. She has been a visiting researcher at SRI International, CA (USA), and George Mason University, VA (USA). She is vice-chair of the IFIP WG 11.3 on Data and Application Security.
<http://www.dti.unimi.it/decapita>



Stefano Paraboschi is a professor and deputy-chair at the Dipartimento di Ingegneria dell'Informazione e Metodi Matematici of the Università degli Studi di Bergamo. He has been a visiting researcher at Stanford University and IBM Almaden, CA (USA), and George Mason University, VA (USA). His research focuses on information security and privacy, Web technology for data intensive applications, XML, information systems, and database technology.
<http://cs.unibg.it/parabosc>



Eros Pedrini is a software engineer at the Department of Information Technology, Università degli Studi di Milano, Italy. He has participated/participates in the EU projects "PRIME" and "PrimeLife". His interests are in the area of information security, privacy, access control, and object oriented languages.
<http://www.dti.unimi.it/pedrini>



Pierangela Samarati is a professor at the Department of Information Technology, Università degli Studi di Milano, Italy. Her main research interests are in data protection, access control, and information privacy and security. She serves as chair and as a member of the steering committees of several conferences. She has served as General Chair, Program Chair, and PC member of various conferences. In 2009, she has been named ACM Distinguished Scientist.
<http://www.dti.unimi.it/samarati>



Mario Verdicchio is an assistant professor at Dipartimento di Ingegneria dell'Informazione e Metodi Matematici of the Università degli Studi di Bergamo. His main interests follow the guidelines and widen the scope of his doctoral thesis on formal semantics of agent communication languages to a more general analysis of formalization of interaction among agents, providing a context for the current research on policy languages for personal data exchange.
<http://cs.unibg.it/verdicch>