

XeNA: an access negotiation framework using XACML

Diala Abi Haidar · Nora Cuppens - Boulahia ·
Frédéric Cuppens · Hervé Debar

Received: 31 March 2008 / Accepted: 13 August 2008 / Published online: 7 October 2008
© Institut TELECOM and Springer-Verlag 2008

Abstract XeNA is a new model for the negotiation of access within an extended eXtensible Access Control Markup Language (XACML) architecture. We bring together trust management through a negotiation process and access control management within the same architecture. The negotiation process based on resource classification methodology occurs before the access control management. A *negotiation module* at the core of this negotiation process is in charge of collecting resources required to establish a level of trust and to insure a successful evaluation of access. The access control management is based on an extended Role-Based Access Control (RBAC) profile of XACML. This extended profile responds to advanced access control requirements and allows the expression of several access control models within XACML.

Keywords Access control · XACML · RBAC · OrBAC · Interoperability · Trust management

1 Introduction

Combining access control and trust is essential to enforce security of Web services and resources shared between entities from different security domains. The

negotiation of access is the first step before the authorization management. It allows the establishment of a required level of trust and collects resources needed in the access request's evaluation process.

In [1–3], authors suggest a prototype for trust establishment called TrustBuilder. It allows negotiating trust across organizational boundaries, between entities from different security domains. The TrustBuilder project is investigating trust negotiation, using an Attribute-Based Access Control (ABAC) [4] model. Using TrustBuilder, parties conduct bilateral and iterative exchanges of policies and certified attributes to negotiate access to system resources including services, roles, capabilities, personal credentials, and sensitive system policies. The TrustBuilder approach consists in gradually disclosing credentials in order to establish trust. The prototype also incorporates policy disclosure; only policies that are relevant to the current negotiation may be disclosed by the concerned parties. These policies specify what combinations of credentials one must present in order to gain access to a protected resource of the accessed service. In this way, it is possible to focus the negotiation and base disclosures on need-to-know requirements. Since these policies may contain sensitive information, their disclosure must also be controlled [5]. Trust may be gradually established, and policies referencing sensitive credentials or containing sensitive constraints are not disclosed to strangers. As far as trust is established, sensitive policies may be disclosed if permitted by the policy.

Regarding access control management, there are many access control models in the literature [TBAC [6], DAC [7], Organization-Based Access Control model (OrBAC) [8], ABAC [4]...], the most frequently cited being the Role-Based Access Control (RBAC) [9]

D. Abi Haidar (✉) · H. Debar
France Telecom R&D Caen, 42 rue des
coutures, 14066 Caen, France
e-mail: diala.abihaidar@telecom-bretagne.eu

N. Cuppens - Boulahia · F. Cuppens
ENST Bretagne, 2 rue de la chataigneraie,
35512 Cesson Sevigne, France

model. Among the languages that have been proposed for expressing access control, there is the eXtensible Access Control Markup Language (XACML), an OASIS standard [10].

In this paper, we bring together negotiation for trust establishment and access control management within the same architecture. Our proposed framework is called XeNA for XACML negotiation of access. Within this framework, we used our methodology of negotiation, based on resource classification, within an extended XACML architecture. In this process, we consider that negotiation precedes the access control evaluation since negotiation allows the collection of resources needed to establish a required level of trust and to permit the success of the access control evaluation.

This paper is organized as follows: In Section 2, we introduce the negotiation's general concept and explain our resource classification. Section 3 presents the negotiation's policies and process. The negotiation's architecture is described in Section 4. The negotiation and exception treatment modules are introduced within this section. In Section 5, we detail our previous work on an extended RBAC profile of XACML on which our architecture is based. Section 6 describes our XeNA framework where the negotiation architecture is integrated within the extended XACML architecture. The implementation of our prototype is discussed in Section 7. Finally, Section 8 concludes the paper and suggests some perspectives.

2 Concept definition

2.1 Access control and negotiation

Every service provider possesses an access control policy that defines who has access to which resource and for which purpose. The Web service is seen as a resource belonging to the service provider. Furthermore, this resource may involve the manipulation of other local protected resources (other services or data). This explains the necessity to determine all the resources that may be concerned by the access request and the definition of their corresponding access control policies. These policies define a set of required prerequisites from the requestor so that the service provider can decide if the access to the requested resource is allowed or not. These prerequisites are attributes obtained from credentials. A *credential* is an *assertion* about its owner, digitally signed by a credential issuer. A credential contains a description of attributes as name/value pairs. An *attribute* is seen as an age, a membership, a job or anything else owned by a person not directly related to

his or her identity. These attributes are used to satisfy access control policies of a requested resource. They are also important in the Trust Establishment (TE) systems [11], where entities tend to trust each other based on attributes.

Access control policies are considered resources belonging to the service provider. However, some policies may not be publicly accessed. In such cases, the required attributes are not necessarily known in advance by the requestor. This latter may not give the required information whenever it accesses the resource. To be able to satisfy the request of legitimate requestors, we propose to negotiate the access trying to collect the needed attributes whenever the need arises.

The negotiation is seen as a process by which a group of participants come to a mutually acceptable agreement on some matter. In a negotiation process [12], there are mainly three components:

1. The negotiation protocols are the set of rules managing the interaction.
2. The negotiation objects are the range of issues over which agreement must be reached.
3. The decision making models are the decision making apparatus the participants employ to act in line with the negotiation protocol in order to achieve their objectives.

According to our reasoning, the primary negotiation object is the access to the Web service seen as a protected resource. This access may involve negotiating other issues such as the access to some protected credentials or policies. That is, policies may be sensitive in the sense that each participant would not wish that the other party knows about his or her access control requirements. We admit that some access control policies or some distillations of these policies, seen as resources, are subject to an access negotiation. In this way, other parties in the negotiation can understand what the requirements are to gain access to the desired resource. The requestor and the provider each implement their negotiation protocols that define how to exchange resources to satisfy the mutual access policies. The decision making model at the level of both entities decides about what to reveal and what to request from the negotiating party according to the negotiation protocol.

2.2 Resource classification

We define as protected resource all sensitive information (e.g., services, sensitive policies, credentials, etc.). All the protected resources in our system are man-

aged by access control policies. We classify protected resources in three classes.

- Class 1—“resource with direct access”: all protected resources that belong to this class are managed by policies that do not trigger a negotiation process. Nevertheless, to evaluate the request for access to such resources, some attributes may be needed (e.g., a requestor identifier). If these attributes are given in the request, the evaluation may be possible. If they are not given, they will not be collected through a negotiation process.
- Class 2—“resource with direct negotiated access”: If the request for access to resources from this class does not include all the needed attributes, the missing attributes are directly requested from the requestor. In a direct negotiation process, the policies are not hidden and may be revealed to the requestor.
- Class 3—“resource with indirect negotiated access”: We consider that the resources that are classified in this class are managed by a policy that should be kept secret. Thus, missing attributes are indirectly requested from the requestor. That is, a strategy should be applied to obfuscate some (or all) of these attributes before negotiating them.

An example of class 1 resources is the *never-accessible* resources. We define these resources as those that cannot be accessed under any circumstance. The resources that are *always accessible* also belong to class 1. The corresponding access control policies allow access to them without restrictions.

As one can notice, the class 2 and class 3 resources may not necessarily imply a negotiation between the requestor and the service provider. Whenever all the needed attributes are given by the requestor directly, there is no need to negotiate. That is, starting a negotiation or not does not reveal to the requestor the classification of the resource. For instance, whenever all the needed attributes are given by the requestor, this latter can imagine that the accessed resource is classified in class 1 since no negotiation was needed whereas, in reality, it could be a class-1, -2, or -3 resource. Similarly, a requestor cannot easily distinguish between class 2 and class 3 resources.

Furthermore, deducing the classification of the resource does not call into question the sensitivity of this resource since our classification is not based on the sensitivity of the resources. That is, a class 1 resource is not necessarily less sensitive than a class 2 (or class 3) resource and vice versa.

2.3 Strategies definition for class 3

We define two types of strategy categories:

- Definition of structured subpolicies: In this category, we consider the strategies based on a step-by-step revealing process. Since the policy that manages the access to a class 3 resource should be kept secret, then other policies (subpolicies) that can be revealed are defined by the administrator. These subpolicies, whenever applied one by one, form a cover for the initial policy. It is only when one subpolicy is verified that the process may continue and additional attributes are requested by the provider.
- Introduction of useless information: We consider the introduction of noise (useless information) as a way of hiding the original policy. That is, additional attributes may be requested from the requestor. These attributes should be given by the requestor even though they are not used in the request's evaluation. If the administrator chooses this kind of strategy, he or she should define the additional useless information to be requested from the user. By using this strategy, we increase the risk of a failure in the negotiation process, i.e., the policies's conditions may not be verified. This occurs whenever the requestor does not give one of the additional required attributes.

We should notice that we do not rule out the possibility of using both strategies at the same time. It is up to the administrator to assign each information that should be hidden to its corresponding strategy.

2.4 Our resources' perimeter of interest

Each service is described by the service provider and its behavior is known in advance. A published Web Service Description Language (WSDL) [13] document describes the interface of the service and the resources that are involved in the proposed service. Nevertheless, a negotiation process may involve other resources that are not directly related to the service; thus, they are not included in its description. These resources may be requested from the provider to satisfy a requestor's policy. Thus, we define three sets of resources (see Fig. 1).

We call organization's resources all the internal resources that belong to an organization (e.g., a provider). Within the set of these resources, some are protected and managed by access control policies since they can be requested by external accesses. They are called the classified resources. Within this set of classified re-

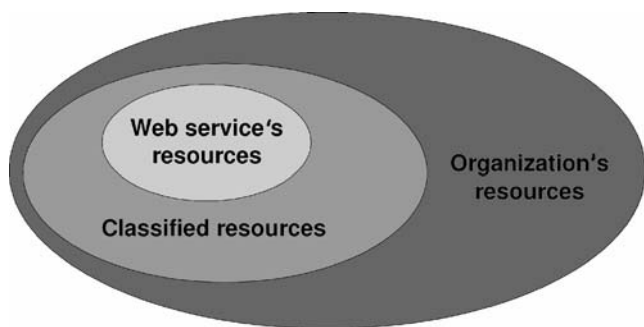


Fig. 1 The organization's resources

sources, we have the resources that are directly related to the Web service's functions.

The resources that are classified and not directly related to the Web service's functions are those that may be involved in the negotiation process. The set of organization's resources can be concerned by another service if we are in the case of a multiservices provider. However, our reasoning concerns a given Web service. We are in the situation where each provider is responsible of one provided service.

3 The negotiation

3.1 Obtaining the negotiation policies

We consider that an access control policy may include information that is strictly related to the organization such as the roles in an RBAC [9] model-based organization. Organizations use such access control models to facilitate the expression and manipulation of their policies. Nevertheless, a local role may not be necessarily interpretable by the requestor. This explains why revealing a local access policy that contains such information is useless. Previous works in the field of interoperability have influenced our choice of exporting a distillation of policies in the negotiation process [5, 14]. In [14], authors make use of the virtual private organizations (VPO) to manage the access control in an interoperability context. Their approach, called O2O (for "Organization to Organization"), is based on the idea that each organization should define a VPO that is associated with security policies managing the access to protected resources within this organization. VPOs manage the interoperability policies of each organization. In the same way, in our architecture, we derive from the access control policies what we call the negotiation policies. These negotiation policies can

be seen as a distillation of the access control policies that is meaningful in the negotiation. The negotiation policies are expressed in a basic way not related to the internal organization and access control model. For this purpose, we suggest translating any defined access control model in an ABAC [4] model. In this way, the negotiation policies state the required attributes to access a resource.

Each access control model using classification of users into roles or actions into activities, etc., bases this classification on attributes. Let us take the example of the RBAC model. The role assignment policies map users to roles. When the user is known in advance to the system, its identity is the only condition for such a mapping, and it is obtained through authentication. When the user is unknown to the system, this mapping is based on attributes not directly related to its identity. Thus, it is necessary to express conditions for mapping users to role, based on attributes (identity or given attributes in a certificate). In the generalized RBAC model [15], objects are classified into "object role." The properties that can be considered are size, creation date, sensitivity level, etc. Thus, the conditions for this mapping into object roles are the attributes of the "object" entity. Moreover, in the OrBAC model [8], actions are grouped into activities based on some properties that can be seen as actions' attributes.

We call *mapping policies* (or *assignment policies*) these policies that define conditions to map users into roles, actions into activities, and objects into object roles.

We define:

$$P : (D, S, A, O, Ctx)$$

$$P_m : (X, group, C)$$

P is an access control policy that is represented by:

- A decision $D \in \{\text{permission, prohibition}\}$
- A group of subjects S
- A group of actions A
- A group of objects O
- Some contextual conditions Ctx that may be relative to the time of access (*temporal conditions*), the place from where the subject sends the request (*spacial conditions*), etc. (see [16] for a classification of different contextual conditions)

If the access control model does not use groups of entities, then S is reduced to a single subject and replaced by s , A is replaced by the action a , and O is replaced by an object o .

P_m is a mapping policy that expresses the conditions C of mapping an entity X into a group *group*.

- X is a subject s , an action a , or an object o .
- *group* is in fact replaced by S , A , or O .
- C is a condition of the mapping. $C \in \{C_s, C_a, C_o\}$, where C_s , C_a , and C_o are the conditions over the attributes of, respectively, the subject, the action, and the object.

If the access control policy P uses the entities s , a , and o , then P is necessarily based on the attributes of these entities. Otherwise, the correlation between the access control policy P and the *mapping policies* (or *assignment policies*) P_m allows the translation of such access control policy into a policy that is only based on attributes. That is, the expression of policies using attributes is always possible whatever the access control model used is. These attribute-based policies are used within the negotiation as *negotiation policies*.

3.2 Negotiation policy properties

The negotiation policies state, according to the accessed resource, the required attributes. One should notice that negotiation policies are only a function of the accessed resource. A negotiation policy applies to a request for access if the accessed resource is managed by this negotiation policy. These policies are modulated by the classification of the resource. In the case of a class-2 resource, the required attributes are asked to the requestor without a specific strategy, whereas, if the concerned resource is in class 3, a strategy should modulate the requested attributes.

Furthermore, the negotiation policies do not express permissions or prohibitions for access. Satisfying a negotiation policy does not imply that the decision for access can be evaluated to permitted or denied. The evaluation of the access is strictly based on the access control policies even though the negotiation policies are derived from the access control policies. To illustrate this distinction, let us take an example in an RBAC-model-based organization. Let us suppose that a resource R can be accessed by a role A and that the role A is assigned to every entity possessing the attribute *Age*, whose value is less than 25. If we consider R as a class-2 resource, the negotiation policy can directly ask for *Age*. When this attribute is received (within a credential) from the requestor, this latter is assigned or not to the role depending on the value of the attribute. The mapping policies evaluate role A 's condition, then the access control policies will grant the access to the requestor if this latter is actually assigned to role A .

3.3 The negotiation process

During the negotiation, multiple rounds may be necessary to satisfy each of the provider and the requestor entity's negotiation policies. Once all the negotiation policies are satisfied, the process may converge to a decision of granting the access or not.

Nevertheless, the negotiation process may not be completed because of some exceptional cases that can be grouped into two categories:

- The nonaccess exceptions are situations where (1) the required resource does not exist at the level of the provider or (2) the evaluation of the request to access a resource leads to a deny decision.
- The loop exceptions are situations where the negotiation process is locked in a loop. Let us suppose that one of the entities involved in the negotiation, E_1 , is waiting for a resource A from the other entity, E_2 . This latter cannot reveal this resource unless E_1 reveals B . If the negotiation policy of B requires A , then E_1 will be asking for A and E_2 for B , and this creates a deadlock situation.

Exceptions are raised after the detection of such situations by the negotiating entities, and an *exception treatment module* is called. This module should propose alternatives so that the negotiation process can be completed. To be able to present our negotiation process and architecture, we should introduce the following definitions.

Definition 3.1 Initial negotiation level (INL): It is a level assigned to an accessed resource at the beginning of its corresponding negotiation process. It reflects the class that the administrator assigns to that resource. The INL of a resource is, thus, the classification of this negotiable resource (INL = 2 or 3).

Definition 3.2 Current negotiation level (CNL): It is the level of a resource within a specific negotiation process. It is initialized to the CNL at the beginning of the negotiation process.

We can imagine that, sometimes, at a given step in the negotiation process, the CNL of the resource can decrease to 1. In fact, whenever the accessed resource is a class-2 or -3 resource, we can imagine that, after satisfying the negotiation policies managing the access to that resource, this latter is seen as a class-1 resource. That is, there are no more needed attributes and the request for access is evaluated.

Property 3.1 *Negotiation termination: The negotiation process is completed if the CNL of the requested resource is decreased to 1.*

When the CNL of the requested resource is decreased to 1, the request's evaluation may conclude (1) on a grant access decision; (2) on a deny access decision; (3) whenever exceptions are raised (because of a denied access or loop), an alternative is found; or (4) the process fails.

We illustrate the negotiation process using state machines. Figure 2 shows the requestor's state machine, the provider state machine is symmetric, by replacing send by receive. The process starts at state 0 with a request for a particular resource. Depending on the classification of this resource on the provider side, the requestor may have direct access to that resource (class 1 resource) or may receive a policy from the provider. The returned policy is directly related to the requested resource (class-2 resource) or it hides, via a strategy, the real policy that manages the access to the resource (class-3 resource). In both cases, the requestor receives a policy and fails to determine if it is a class-2 or class-3 resource. The returned policy asking for some attributes is evaluated by the requestor. If the requestor found credentials certifying such attributes, it sends them to the provider (if they are class-1 resources). The requestor returns to state 1 since we cannot know if the accessed resource's policy is satisfied or if other exchanges of policies are required. From state 2, the requestor has the possibility also to return a policy that manages the found credentials (class-2 or -3 resources). It is in state 3 and waits for the provider's credentials. In this state, it may directly receive credentials from the provider and go to state 1, or it may receive a policy

and go back to state 2. If it goes to state 1 after state 2, it still has to send credentials to satisfy the received policy from the provider. Finally, after receiving the requested resource, it goes to state 0. To express the exceptional cases, the state machine of the requestor and the provider should be enriched (state 4 in Fig. 2).

4 The negotiation architecture

We have defined a basic architecture that shows the main actors within the negotiation process. Each entity should have a classification of its own sensitive resources and its specific access control policies to protect these resources. It should also have negotiation policies. We have defined, for each entity involved in the process, a specific module that is in charge of the negotiation process. This *negotiation module* uses the negotiation policies to negotiate access to the protected resources. If an exception is raised, this module calls the *exception treatment module* so that it checks the exception treatment policies to find if there is a possible alternative. The global data-flow is presented in Fig. 3.

Each request for an access is intercepted by the negotiation module. If the requested resource is classified in class 1, the request is relayed for evaluation on the service provider side. If the resource is classified in class 2 or 3, the negotiation module will start the negotiation. The resource's INL (=CNL) within this negotiation process is equal to 2 or 3 (depending on the class of the resource). The negotiation module checks what the accessed resource's negotiation policies are. This module ensures that all the required attributes are collected, and then the CNL is assigned the value 1. At this level in the negotiation process, the request with the collected attributes is sent to the service's evaluation point.

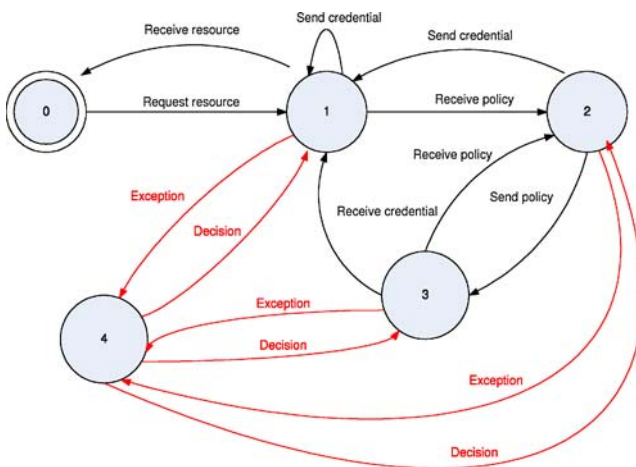


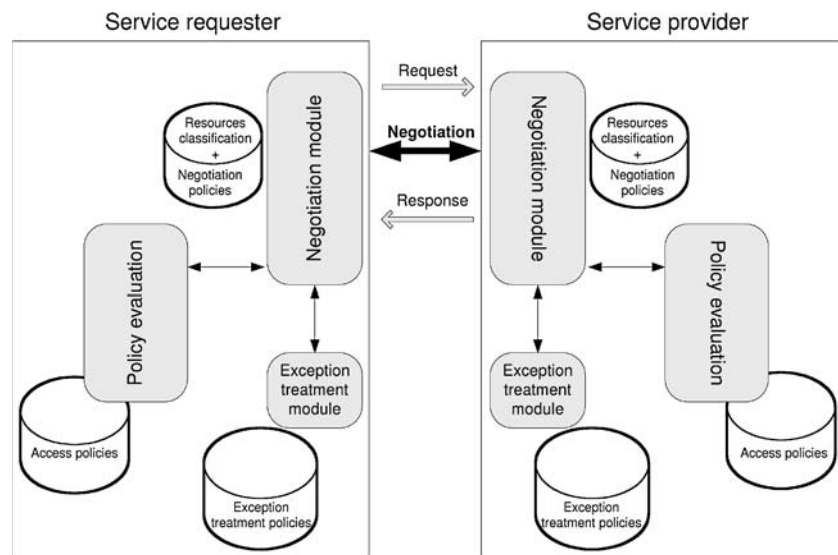
Fig. 2 Requestor's negotiation process state machine

4.1 The exception treatment module

Exceptions are caught by an *exception treatment module*, which evaluates alternatives the administrator has defined for each exception case. However, who will solve the exception: is it the requestor, the provider or both? Actually, we can consider that each entity chooses to react or not depending on the strategy defined to manage exceptions. These strategies may be:

- Strategy 1: Abort the negotiation without solving the exception.
- Strategy 2: Solve the exception if possible, independently of the actions undertaken by the other entity.

Fig. 3 The negotiation global architecture



- Strategy 3: Wait a certain amount of time for the other entity to solve the exception, then return to strategy 1 or 2.

According to the chosen strategy, the administrator should define some policies for solving the exceptions. The treatment module will check these policies; if there is an exception treatment policy, then it is applied. If not, the negotiation will abort.

When an exception is caused by a loop, it should be detected by the two entities involved in the negotiation process so that one can solve it. In the case of nonaccess exception, this exception should also be raised by the two sides even though, for the provider, the evaluation of the access policy may have terminated on a deny decision, which means that there was no problem. We should always take into consideration these exceptions because the service requestor may propose another alternative (another credential for instance). Thus, it is essential not to abort the negotiation and initialize another one when the requestor submits an alternative. In this case, one obvious strategy for the provider is strategy 3.

4.2 The negotiation module

The negotiation module (Fig. 4) is split in two parts: the *supervision layer*, which is responsible for the supervision tasks (see Section 4.2.1), and the *execution layer*, which enforces the negotiation protocol according to the directives of the supervision layer (see Section 4.2.2). This abstraction is applied whatever the underlying implementation is.

4.2.1 The supervision layer

The supervision layer has the following functions:

1. Checking the classification of the resource: The negotiation module uses a repository of the classes to which all the classified resources within the organization belong. Each time the request for access to a resource is intercepted by the module, this latter looks at its class. If the resource is in class 1, then the request is sent to the evaluation point. Whenever its class is 2 or 3, the negotiation module checks the *negotiation policies repository*.
2. Checking the negotiation policies repository: This repository contains all the negotiation policies. Since each negotiation policy manages the access to a specific resource, the policies are selected within this repository according to the accessed resource.
3. Decreasing the CNL of the accessed resource: The negotiation module should take into account the

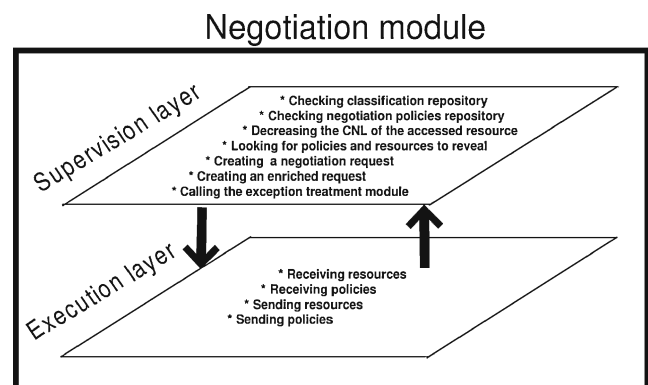


Fig. 4 The negotiation module’s architecture

classification in the negotiation process. Each time the required attributes are collected and the resource's negotiation policies are satisfied, the CNL of the accessed resource is decreased to 1.

4. Looking for policies and credentials to reveal: The module is able to reveal some negotiation policies and some credentials to satisfy the requestor's negotiation policies. To do that, it analyzes a received negotiation policy and determines the local credentials that can be revealed. It may also determine what the negotiation policies (if any) that should be revealed to the requestor are. We consider that only negotiation policies (not access control policies) can be revealed. This is because the access control policies may be expressed using a group of entities that cannot be interpreted by the other entity.
5. Creating an enriched request: Each time the negotiation module collects credentials certifying some attributes that are necessary for evaluating a request, it forms an enriched request where it brings together all the collected information.
6. Creating a negotiation request: After finding the local credentials that satisfy a received negotiation policy, the negotiation module should determine if these credentials are accessible or not. For this purpose, it makes a negotiation query to access such credentials. This negotiation query is treated as any query for access to a resource.
7. Calling the exception treatment module: Whenever an exception is detected by the negotiation module, this latter calls the exception treatment module and sends the raised exception. The exception treatment module checks if there is any alternative. As stated before, the exception treatment module has some predefined policies that handle different cases of exceptions and suggest alternatives.

4.2.2 The execution layer

The execution layer is in charge of executing the tasks of the supervision layer. It also receives information from the execution layer of another negotiating entity. There is an execution layer at the level of the requestor, but there is also one at the level of the provider. The tasks of the execution layer are:

1. Receiving resources: The execution layer of the requestor receives resources revealed by the provider. These resources are relayed to the local supervision layer.
2. Receiving policies: If the provider requests attributes to be able to reveal some of its resources,

the execution layer of the requestor receives the corresponding policies to be enforced. These policies are then relayed to the supervision layer, which can determine what the credentials that enforce these policies are.

3. Sending policies: Whenever the requested resources are classified in classes 2 or 3, policies are revealed by the provider before giving access to these resources. The supervision layer determines the policies that can be revealed and passes them to the execution layer. This latter sends these policies to the requestor.
4. Sending resources: If the supervision layer decides that the requested resources (or credentials) can be revealed, it relays them to the execution layer so that this latter sends them to the requestor.

Now that we have presented our resource classification-based negotiation approach and architecture, we will discuss how we fit the negotiation into an authorization standard such as XACML. The use of an existing standard has the benefit of promoting the interoperability and reducing the efforts required to integrate with existing applications. The following section introduces the extended RBAC profile of XACML on which we base the negotiation.

5 An extended RBAC profile of XACML

5.1 A brief introduction to XACML

XACML is an OASIS standard [10]. It is expressed in XML and describes both a policy language and an access control decision request/response language. The policy language is used to describe general access control requirements to resources in the information system. The request/response language allows one to form a query to ask whether or not a given action should be allowed, and the response will convey the answer for this query. The answer should contain one of these four values: permit (access allowed), deny (access denied), indeterminate (an error occurred or some required value was missing, so a decision cannot be made), or not applicable (this service has no policy that applies to this request). The OASIS standard defines a data-flow diagram (see the Fig. 5). The typical setup is that someone or a process wants to perform some action on a resource. Therefore, a request is sent (2) to the component that actually protects that resource (like a filesystem or a Web server), called a Policy Enforcement Point (PEP). The PEP will form a request, in its native request format, based on the requestor's

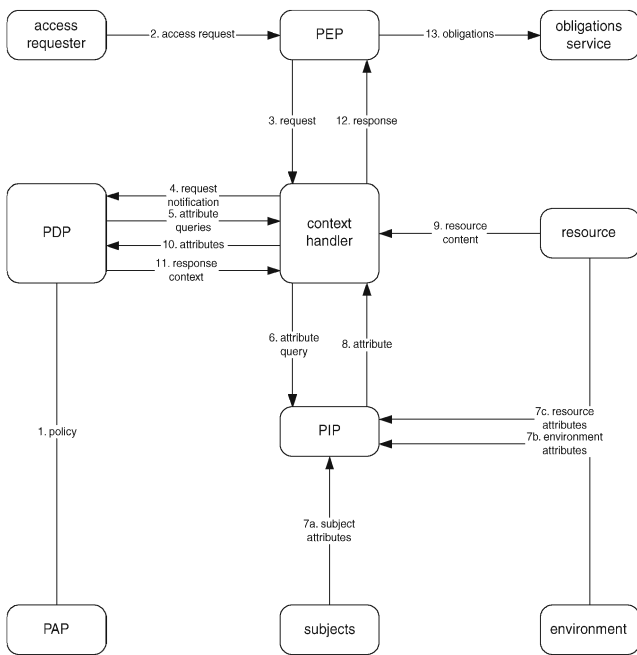


Fig. 5 The OASIS XACML data-flow diagram

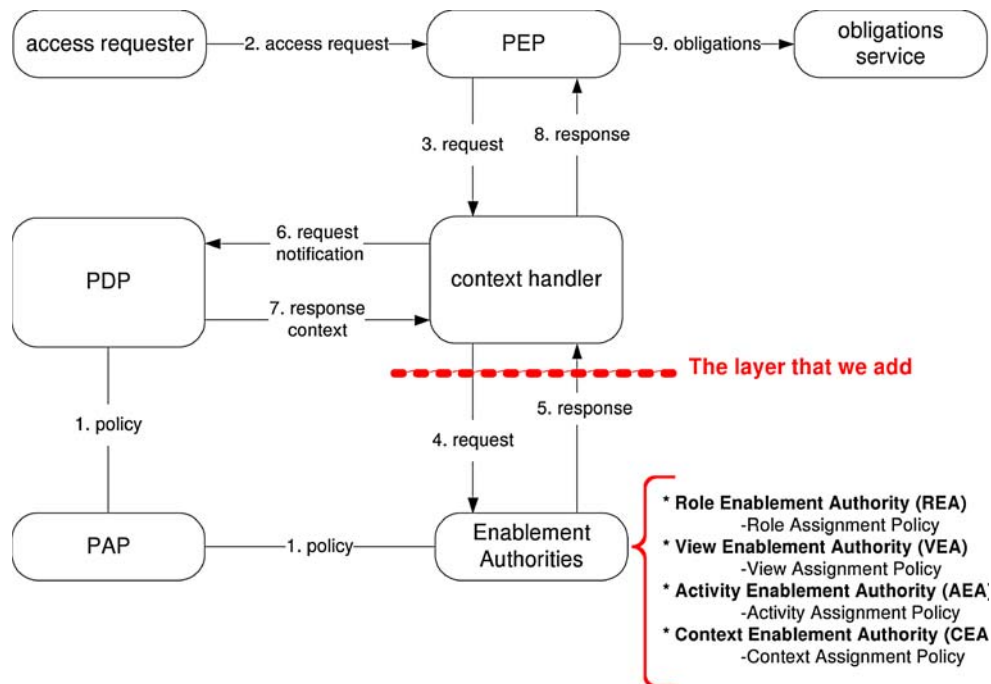
attributes, the resource in question, the action, and other information pertaining to the request. This request is sent to the context handler (3) that constructs an XACML request context for a Policy Decision Point (PDP) (4), which will look at the request and some policy that applies to the request. The policies have been written by the Policy Administration Point (PAP)

and made available to the PDP (1). Sometimes, the PDP may require additional attributes while evaluating the request. In this case, attribute queries are sent to the context handler (5); this latter requests the attributes from the Policy Information Point (PIP) (6) then (7, 8, 9) sends them back to the PDP (10). The PDP will finally evaluate the policy and come up with an answer about whether access should be granted (11). That answer is returned to the PEP via the context handler that translates it to the native response format of the PEP (12). The PEP can then allow or deny access to the requestor and possibly fulfill some obligations (13).

5.2 The extended RBAC profile of XACML

OASIS has defined profiles for XACML that address various requirements in the access control field. The Security Assertion Markup Language (SAML) profile of XACML [17] provides means to write assertions regarding the identity, attributes, and entitlements of a subject, and defines a protocol for exchanging these assertions between entities. The core and hierarchical RBAC profile of XACML [18] defines a profile for the use of XACML to meet the requirements of RBAC [9, 19]. In previous work [20], we have proposed an extended RBAC profile of XACML with an extended architecture for the XACML architecture (see the Fig. 6). This extended profile responds to more advanced access control requirements such as user–user

Fig. 6 The extended profile architecture



delegation, access elements abstractions, and contextual applicability of the policies. Every access request is composed of a user asking if his/her action over a given resource is allowed or not. In our profile, we have used the concrete entities of the request (subject, action, and object), but, following the OrBAC model, we have also defined the abstract entities (roles, activities, and views) in addition to the context. We have chosen the following terminology for the abstract entities: *view* and *activity* are used to classify the objects and actions, respectively, to which the same security rules apply and *role* is kept as abstraction of subjects. The *context* is used to express the contextual information (temporal, spatial, provisional, prerequisite, and user-declared [8]). These different abstract entities are managed in our architecture by different Enablement Authorities (EA) that are in charge of the translation between the concrete entities of the request and the abstract ones. There are four different EA for managing role (REA), view (VEA), activity (AEA), and context (CEA), respectively. Each EA can query the corresponding assignment policy. We consider that

there is a Role Assignment Policy (RAP) that defines which roles are assigned to a given subject. The REA maintains a list of all the roles that are defined in the organization, and when asked about the role that can be assigned to a subject, the REA makes a request to the RAP to get the set of candidate roles. The VEA can similarly make requests to a view assignment policy to determine to which view(s) the accessed object belongs. The AEA similarly manages activity assigned to actions via an activity assignment policy. Finally, the contextual conditions are taken into consideration by the CEA. The new data flow is presented in Fig. 6.

The policies are written by the PAP (1). When the PEP receives a concrete request (2), it is relayed to the context handler (3). The context handler then collects the assignments of the requested subject to its roles, the requested resource to its views, and the requested action to its activities and finally gets the value of the contexts. This is done by sending requests to the corresponding EA (4). Once all the responses have been collected (5), the context handler sends a request to the PDP for evaluation (6). This request contains the

Listing 1 An example to illustrate the new attributes

```

1  <?xml version="1.0" encoding="UTF-8"? >
2  <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
5      http://docs.oasis-open.org/xacml/2.0/
6      access_control-xacml-2.0-context-schema-os.xsd" >
7  <Subject>
8  <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
9      DataType="http://www.w3.org/2001/XMLSchema#anyURI" >
10     <AttributeValue>urn:example:role-values:physician</AttributeValue>
11   </Attribute>
12 </Subject>
13 <Resource>
14 <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:resource:view"
15     DataType="http://www.w3.org/2001/XMLSchema#anyURI" >
16   <AttributeValue>urn:example:view-values:medical-file</AttributeValue>
17 </Attribute>
18 </Resource>
19 <Action>
20 <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:action:activity"
21     DataType="http://www.w3.org/2001/XMLSchema#anyURI" >
22   <AttributeValue>urn:example:activity-values:checking</AttributeValue>
23 </Attribute>
24 </Action>
25 <Environment>
26 <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:environment:context"
27     DataType="http://www.w3.org/2001/XMLSchema#anyURI" >
28   <AttributeValue>urn:example:environment-values:designated_doctor
29 </AttributeValue>
30 </Attribute>
31 </Environment>
32 </Request>

```

abstract values and the concrete values of the subject, action, and object of the initial request in addition to the value of the context(s). The PDP evaluates the request according to the policies and sends back a response to the context handler (7) that will transmit it to the PEP in its native response language (8). The PEP may have to fulfill some obligations (9) before allowing or denying the access.

5.2.1 Used XACML attributes

We have extended the XACML language with new attributes to manage the abstract entities that we have considered (role, view, and activity) and context. New attributes have been defined to handle the role values (XACML subject attributes); the view values (XACML resource attributes); the activity values (XACML action attributes); and, finally, the context values (XACML environment attributes).

Listing 1 shows an illustration of these extensions. It is an example of an XACML request the context handler may send to the PDP for evaluation. On lines

8, 14, 20, and 26, we can see the identifier attribute of our new attributes.

5.2.2 Assignment policies

Each EA queries the assignment policy it manages, asking whether a given concrete entity can be assigned to one of the defined abstract entities. As such, the EA determine the value of the entity they have in charge (i.e., role value, activity value, view value, and context value) that matches the initial request entities (i.e., subject, action, and object). The response of the evaluation against the assignment policies can be *Permit*, *Deny*, *NotApplicable*, or *Indeterminate* for each request. The response *permit* means that the corresponding value is assigned.

Listing 2 illustrates an example of an activity assignment policy. In this example, the unique rule will match a request sent by the AEA asking whether a subject, the action *read* (line 9), can perform the action *enableActivity* (line 25) over the resource the activity *checking* (line 17). In other words, this rule states that

Listing 2 An example of an activity assignment policy

```

1 <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" PolicyId="Activity:Assignment:Policy"
2   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
3   <Target/>
4   <Rule RuleId="checking:activity:requirements" Effect="Permit">
5     <Target>
6       <Subjects>
7         <Subject>
8           <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
9             <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
10            <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
11              DataType="http://www.w3.org/2001/XMLSchema#string"/>
12            </SubjectMatch>
13          </Subject>
14        </Subjects>
15        <Resources>
16          <Resource>
17            <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
18              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
19                urn:example:activity-values:checking
20              </AttributeValue>
21              <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:action:activity"
22                DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
23            </ResourceMatch>
24          </Resource>
25        </Resources>
26        <Actions>
27          <Action>
28            <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
29              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
30                urn:oasis:names:tc:xacml:2.0:actions:enableActivity
31              </AttributeValue>
32              <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
33                DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
34            </ActionMatch>
35          </Action>
36        </Actions>
37      </Target>
38    </Rule>
39  </Policy>

```

it is permitted to assign the action *read* to the activity *checking*.

6 Negotiation within XACML architecture

We have used the resource classification-based negotiation architecture and the extended RBAC profile to define a global negotiation framework based on the XACML architecture. We call this XACML-based negotiation of access framework XeNA. Within XeNA, the negotiation is considered first to collect all the necessary attributes that permit to increase the chance of a successful request evaluation.

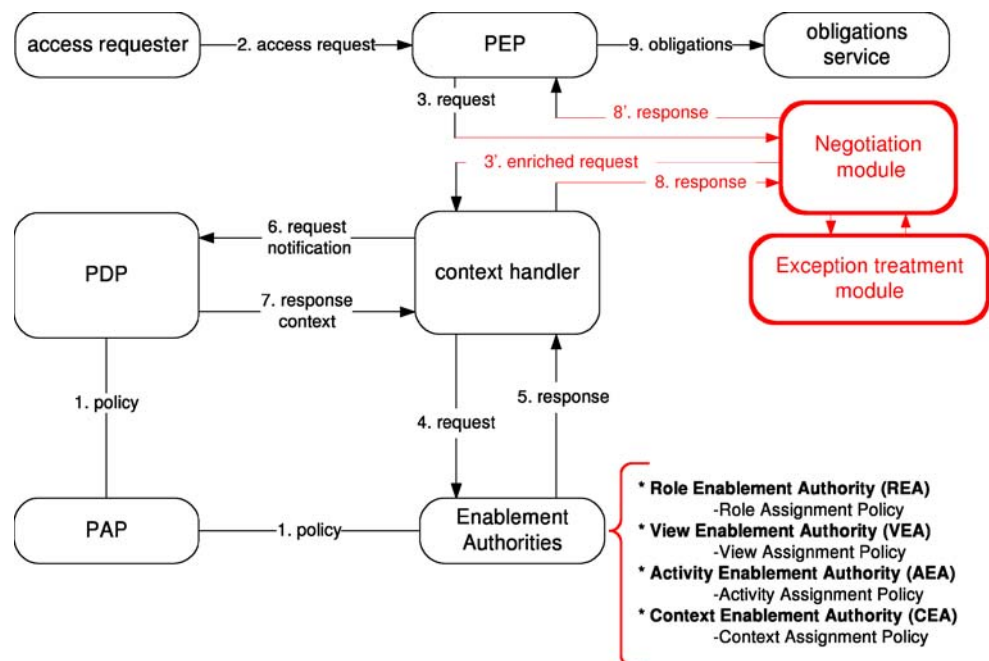
6.1 XeNA: an extended negotiation architecture

The data-flow in this architecture will be as follows (see Fig. 7): After an access request arrives at the PEP (2), this latter relays it to the context handler. At this level, the negotiation module acts as a hook; it intercepts all the requests that are intended for the context handler (3). If the requested resource is a class 1 resource, the negotiation module sends the request to the context handler and the data-flow will be the same as the one defined in our extended profile (see Fig. 6). If the requested resource is classified in class 2 or 3, then the negotiation module will start the negotiation after checking the defined negotiation policies made available to it by the administrator. The negotiation will continue until the CNL of the requested resource is 1

(see Section 3.3). At this point, the request enriched with the collected information is relayed to the context handler (3'). The context handler executes the normal actions (4, 5, 6), and the evaluation of the request is done at the level of the PDP. The PDP sends the evaluation result (permit or deny) to the context handler (7); this latter sends it back to the PEP (8). The negotiation module also intercepts this response to check if there is a nonaccess exception. In this case, the exception treatment module is called. Afterwards, the negotiation module sends the response back to the PEP (8'), which applies the decision.

Until now, we did not take into consideration the credentials certifying the attributes that the other entity involved in the negotiation process may request. In fact, we consider that the negotiation module is in charge of negotiating the access by collecting credentials and revealing policies. It is up to the negotiation module to submit an access request concerning the required credentials to the context handler. This request is similar to any request received from the PEP. The subject of the request should be the other negotiating entity, and the requested resource is the credential that contains the attributes required within the negotiation process. If the access policies evaluated by the PDP permit the access to that resource, it will be revealed so that the other entity gives the initial requested resource and the negotiation continues. Elsewhere, an alternative should be proposed by the exception treatment module so that the negotiation module may continue the negotiation.

Fig. 7 Data-flow in the new architecture



6.2 Failures in the negotiation

Regarding failures in the negotiation, several cases may be considered:

- Case 1: The negotiation module checks the class of the resource (negotiable resource) and does not find the corresponding negotiation policy that applies to that resource. It cannot launch the negotiation process.
- Case 2: The negotiation module fails during the negotiation process while collecting resources because of a loop exception raised without alternative.
- Case 3: The negotiation module fails during the negotiation process while collecting the required resources because of a nonaccess exception raised by the requestor or the provider without alternative.
- Case 4: The negotiation module contacts the exception treatment module because there is a final nonaccess exception. This occurs after the access decision has been already evaluated by the PDP.

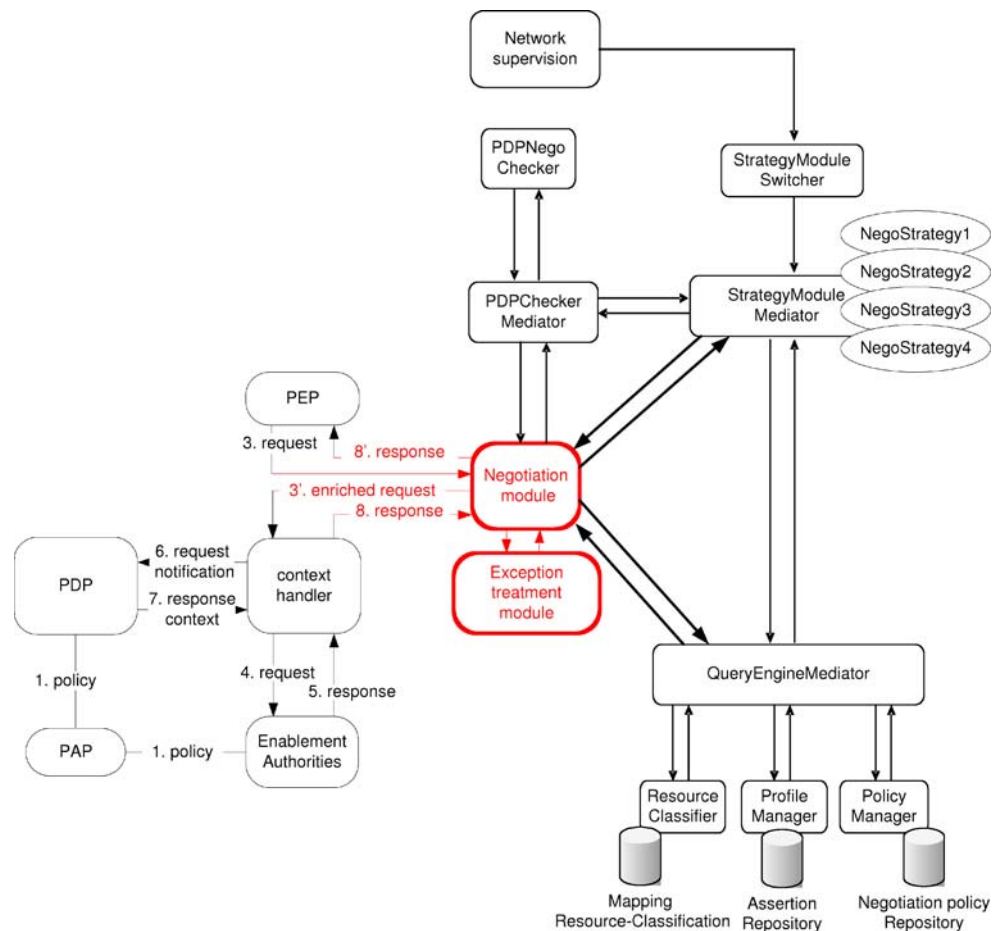
The exception treatment module has no alternative to suggest.

In cases 1, 2, and 3, there is a failure in the bottom-up flow (from the PEP to the PDP in Fig. 7). The INL of the resource is 2 or 3 and it is not possible to derive a CNL of 1; then, the negotiation module should not relay the request to the context handler. The needed attributes are not collected; thus, the request for access cannot be properly evaluated. If an alternative (e.g., other combination of requested attributes) were proposed by the exception treatment module, the negotiation would have continued or, in the absence of alternatives, the access is denied. In case 4, it is a failure in the top-down flow (from the PDP to the PEP in the Fig. 7); then, the response is sent to the PEP so that it applies the decision.

7 Implementation

The implementation of the RBAC profile of XACML does not yet exist; however, the XACML standard

Fig. 8 The implementation of the negotiation functionalities



specification version 1.1 is implemented by Sun.¹ Such an implementation especially provides the functionalities of a PDP for evaluating a request against an XACML policy. According to our profile, the EA are based on such XACML PDP functionalities in the sense that they evaluate XACML requests for enablement against existing assignment policies. The context handler and PEP specified in the XACML standard [10] are not implemented in Sun's distribution. We have implemented such entities and made some modifications to the existing Sun's implementation in order to respond to the requirements of our extended RBAC profile of XACML. That is, the environment value, as defined in the XACML standard, is never evaluated in the existing Sun's implementation. This environment information is needed in our extended profile in order to evaluate the context value.

In the XeNA framework, the negotiation architecture is integrated within the implemented extended RBAC profile of XACML (see Fig. 8). We based the negotiation architecture implementation on the open source distribution code of TrustBuilder2 version 0.1 [21]. This distributed framework for trust negotiation can be easily extended by adding various plugins that can be loaded by the TrustBuilder2 runtime. These plugins can be used in place of, or in addition to, the system components. We have modified many classes of the TrustBuilder2 prototype to express XeNA requirements. That is, the actual distribution of TrustBuilder2 supports a policy compliance checker based on CLOUSEAU policy language. However, TrustBuilder2 supports any type of policy language supposing that this language is added to the system by implementing a derived class of *AbstractPolicyBrick*, i.e., the abstract policy type supported by TrustBuilder2. We have used XACML to express access control and negotiation policies. Consequently, we implemented a new compliance checker based on the functionalities of an XACML PDP. It verifies some conditions against the existing or received credentials (i.e., embedded attributes). We have developed *ExceptionTreatmentModule*, *NegotiationModule*, and *ResourceClassification* classes and many others.

We needed a module that returns the negotiation policies and resources' classification upon request. We have used the *QueryEngineMediator* module of TrustBuilder2 that dispatches classes to appropriate query

engines based upon their query type. Currently, this module supports two query engines in TrustBuilder2: the *PolicyManager*, responsible for loading policies, and the *ProfileManager*, which loads credentials and claims (i.e., uncertified information such as phone numbers or e-mail addresses). We have added the *ResourceClassifier* engine, which loads resource classification. The *PolicyManager* engine is exclusively dedicated to the negotiation policies in our case.

The actual *StrategyModuleMediator* in TrustBuilder2 is used to call a strategy that was chosen for use during a given session. This module decides what should be disclosed in a response to the other negotiating party. The current module supports one trust negotiation strategy. We have extended this module in order for it to be able to choose but also dynamically change strategies during the negotiation process. Four classes, each implementing a different negotiation strategy, were created.

The exchanged credentials are based on the SAML V2.0 standard [22]. In order to implement SAML credentials, we have used some open source Java libraries of the openSAML2.0.² Finally, we tested our XeNA prototype according to some defined test cases, and we obtain the expected results.

8 Conclusion and perspectives

In this paper, we have proposed a framework for access negotiation and access control management. The negotiation of access aims to establish trust and collect data necessary for a successful access evaluation. This negotiation is processed by a *negotiation module* and based on a resource classification methodology. The access control management is done within an extended RBAC profile of XACML. Our proposition is based on this extended profile architecture, to which we add the negotiation actors (negotiation module and exception treatment modules).

Now that we have proposed a structured framework for negotiation within XACML, further work should be done on the way policies can be evaluated according to the resources. This is to determine what the required resources to satisfy these policies are. We will also improve our implemented prototype especially by reducing the time it takes for a negotiation process to complete.

¹<http://sunxacml.sourceforge.net/>

²<https://spaces.internet2.edu/display/OpenSAML/Home>

Acknowledgements The work presented in this paper is supported by funding from the ANR RNRT POLITESS Project.

References

1. Seamons KE, Chan T, Child E, Halcrow M, Hess A, Holt J, Jacobson J, Jarvis R, Patty A, Smith B, Sundelin T, Yu L (2003) TrustBuilder: negotiating trust in dynamic coalitions. In: Proceedings DARPA information survivability conference and exposition, vol 2. Washington, DC, pp 49–51, 22–24 April 2003
2. Smith B, Seamons KE, Jones MD (2004) Responding to policies at runtime in TrustBuilder. In: Proceedings of the fifth IEEE international workshop on policies for distributed systems and networks (POLICY'04). New York, pp 149–158, 7–9 June 2004
3. Seamons KE, Winslett M, Yu T, Chan T, Child E, Halcrow M, Hess A, Holt J, Jacobson J, Jarvis R, Patty A, Smith B, Sundelin T, Yu L (2003) Trust negotiation in dynamic coalitions. In: Proceedings of the DARPA information survivability conference and exposition (DISCEX'03), vol 2. Washington, DC, pp 240–245, 22–24 April 2003
4. Yuan E, Tong J (2005) Attribute based access control (ABAC): a new access control approach for service oriented architectures. In: Ottawa new challenges for access control workshop. Ottawa, April 2005
5. Seamons K, Winslett M, Yu T (2001) Limiting the disclosure of access control policies during automated trust negotiation. In: Network and distributed system security symposium. San Diego, pp 45–56, April 2001
6. Thomas RK, Sandhu RS (1997) Task-based Authorization Controls(TBAC): a family of models for active and enterprise-oriented authorization management. In: Proceedings of the IFIP WG11.3 workshop on database security. Lake Tahoe, pp 166–181, August 1997
7. Harrison MA, Ruzzo ML, Ullman JD (1976) Protection in operating systems. *Commun ACM* 19(8):461–471
8. Mieke A (2005) Definition of a formal framework for specifying security policies. The Or-BAC model and extensions. PhD thesis, ENST, June
9. Sandhu RS, Coyne EJ, Feinstein HL, Youman CE (1996) Role-based access control models. *Computer* 29(2):38–47, February
10. eXtensible Access Control Markup Language (XACML) Version 2 (2005) Standard. OASIS, February
11. Herzberg A, Mass Y, Michaeli J, Ravid Y, Naor D (2000) Access control meets public key infrastructure, or: assigning roles to strangers. In: SP '00: proceedings of the 2000 IEEE symposium on security and privacy. IEEE Computer Society, Washington, DC, p 2
12. Jennings NR, Faratin P, Lomuscio AR, Parsons S, Wooldridge MJ, Sierra C (2001) Automated negotiation: Prospects, methods and challenges. *Group Decis Negot* 10(2):199–215
13. Christensen E, Curbera F, Meredith G, Weerawarana S (2001) Web Services Description Language (WSDL) 1.1. W3C note, Microsoft and IBM Research, March
14. Cuppens F, Cuppens-Boulahia N, Coma C (2006) O2O: virtual private organizations to manage security policy interoperability. In: ICISS. Delhi, pp 101–115, 17–21 December 2006
15. Moyer MJ, Abamad M (2001) Generalized role-based access control. In: 21st International conference on distributed computing systems. Mesa, pp 391–398, 16–19 April 2001
16. Cuppens F, Miège A (2003) Modelling contexts in the Or-BAC model. In: ACSAC. Las Vegas, p 416, 8–12 December 2003
17. SAML 2.0 profile of XACML v2.0 (2005) Standard. OASIS, February
18. Core and hierarchical role based access control (RBAC) profile of XACML v2.0 (2005) Standard. OASIS, February
19. Ferraiolo DF, Sandhu R, Gavrila S, Kuhn DR, Chandramouli R (2001) Proposed NIST standard for role-based access control. *ACM Trans Inf Syst Secur (TISSEC)* 4(3):224–274
20. Haidar DA, Cuppens-Boulahia N, Cuppens F, Debar H (2006) An extended RBAC profile of XACML. In: Proceedings of 2006 ACM secure web services workshop (SWS). Fairfax, pp 13–22, November 2006
21. Lee AJ (2007) TrustBuilder2 user manual version 0.1. Technical report, May
22. Security Assertion Markup Language (SAML) V2.0 Technical Overview (2006) Working draft 10. OASIS, October