Modern Standard-based Access Control in Network Services: XACML in action

Piervito G. Scaglioso, Cataldo Basile, Antonio Lioy

Politecnico di Torino, Torino, Italy

Summary
Access control in distributed systems is a complex problem that can be tackled in several ways. The XACML standard provides a possible solution, with several benefits and some drawbacks. In this paper we investigate the concepts behind distributed access control, review the XACML standard, and provide practical suggestions about the components to be used in building a XACML-based distributed access control system.

Key words:
Security Policy, Authorization, Access Control, XACML.

1. Introduction

Access control is the ability to permit or deny to a specific subject the use of a resource.
In a general scenario the access control process is managed by an authorization system (AS) that takes decisions according to some authorization policies.
An authorization policy consists of a set of 4-tuples *(s,r,a,c)* stating that the subject *s* is allowed to perform the action *a* on resource *r* if the set of condition *c* evaluates to true. The set of actions a subject is allowed to perform on a resource are called privileges.
Authorization policies are implemented on resources by ACA (access control agents). An ACA sends to the AS a request as a triple (*s,r,a*) asking if the subject *s* owns the necessary privileges to perform action *a* on resource *r*. For example, in the access request "Allow the finance manager to create a file in the invoice folder on the finance server," the subject is the "finance manager," the resource is the "invoice folder on the finance server," and the action is "create a file."
In a traditional centralized computing architecture (based upon mainframe and terminals) the role of AS and ACA was mainly played by the operating system (OS). Multi-user OS employ various security features (e.g. password-based authentication, file-based Access Control List known as ACL) to identify users and to permit or deny actions. Additionally, the OS is able to log all the security relevant events. The assumption was that users interact with a centralized system through devices having no autonomous computational capability, being mere I/O devices (i.e. pure terminals). Therefore all actions are performed on the hardware controlled by the OS which is then able to maintain a complete and consistent view of the system's state and to enforce privileges and restrictions.
Nowadays this monolithic and centralized approach to access control is largely infeasible because modern computing paradigms heavily exploit the concept of distributed computing. We are surrounded by a growing number of interconnected entities having increasing computational power. Peer-to-peer protocols are gaining ground and already substituted the traditional client-server approach in various fields (e.g., bit torrent[1]). To better use resources, grid computing [2] split up programs into parts that run simultaneously on multiple computers communicating over a network Computers may also interact without human intervention, for example using the Web Service technology. Moreover, mobile software agents can choose to migrate between computers at any time during their execution [3]. It is therefore clear that the access control problem in distributed systems cannot be simply solved by the OS because there is no way to keep a complete and consistent view of the global state of the system. For this reason, many alternative access-control mechanisms for distributed systems have been proposed in the literature. The proliferation of independent solutions led researchers to concentrate on more general schemes and authorization frameworks (e.g., Akenti [4], Ponder [5], WS-Policy [6], PERMIS [7]).
All these frameworks specify their own policy languages, enforcement technique, and data formats. It is evident that a common standardized method for access control and policy enforcement it absolutely needed to build interoperable distributed systems: first of all, because the administration of different systems may require the usage of various access control methods, and second because cooperation between different security domains (e.g., merging policies from two different interacting companies) would otherwise become a nightmare.
XACML [8] (eXtensible Access Control Markup Language) is the standard proposed by OASIS (the Organization for the Advancement of Structured Information Standards) to simplify these problems.

Manuscript received December 5, 2008
Manuscript revised December 20, 2008

XACML was designed to replace existing access control mechanisms. It makes possible a simple, flexible way to express and enforce access control policies in a variety of environments, using a single language. It has a number of advantages over other access control policy technologies. For instance, security administrators can describe an access-control policy once, without having to rewrite it several times in different application-specific languages. On the other hand, application developers don't have to invent their own policy language and write code to support it because they can reuse existing standardized code.
Theoretically, XACML is the definitive solution to access control problems in a distributed scenario. However its practical use opens several issues. In this paper we investigate the concepts behind distributed access control, review the XACML standard, and provide practical suggestions about the components to be used in building a XACML-based distributed access control system.
The paper is organized as follows. Section 2 provides a clear overview of the XACML standard. After that, in Section 3 we present more in details the XACML entities interactions. Section 4 analyses some of the most interesting XACML implementations, looking towards their application to the web services and the Service Oriented Architecture (SOA) [9]. Section 5 presents the actual interoperability between XACML entities, highlighting the open issues that according our opinion, should be take into account to improve XACML. Section 6 gives the conclusions.

2. An overview of XACML

XACML is an XML-based syntax that describes both a policy language to specify general access control requirements and request/response formats for the authorization process.
It provides:

- a way to base access control decisions on attributes of both a subject and a resource;
- a mechanism for supporting multiple subjects who have multiple roles (addressed by the XACML profile for RBAC [10]);
- a method to share policies in a distributed environment;
- a way to separate policy definition from its implementation in the applications.

Additionally XACML suggests a management architecture for the decision-making process. This architecture is described by a data-flow model based on the ITU Recommendation X.812 [11] and on the standard ISO/IEC 10181-7 [12][13].
It is worth noting that while the formats are standard and defined by means of an XML Schema, the usage of proposed architecture is not mandatory and does not constitute a standard.

In the next sections we will present basic properties of XACML language and architecture.

2.1 The XACML Policy language

The policy language model is composed of several hierarchical objects depicted in Figure 1.
XACML policies are XML documents rooted in a Policy or PolicySet element. A PolicySet is a container that can hold other Policy or PolicySet instances.
A Policy represents a single access control object, expressed through a set of Rule elements.
A Rule is composed by one Target, one or more Conditions and an Effect.
A Target is basically a set of simplified attribute values to uniquely identify Subject, Resource and Action. For example, a username, their group membership, the file they want to access, and the time of day are all attribute values.
Policy and PolicySet may be associated to Target elements. The Target is used both to check the request applicability and to index the Rule, Policy and PolicySet.
The Condition is an optional Boolean function used to further refine the applicability of the Rule. For example in the sentence "Only allow logins from 9am to 5pm" the conditions serves to indicate the [9am-5pm] interval in which access must be granted.

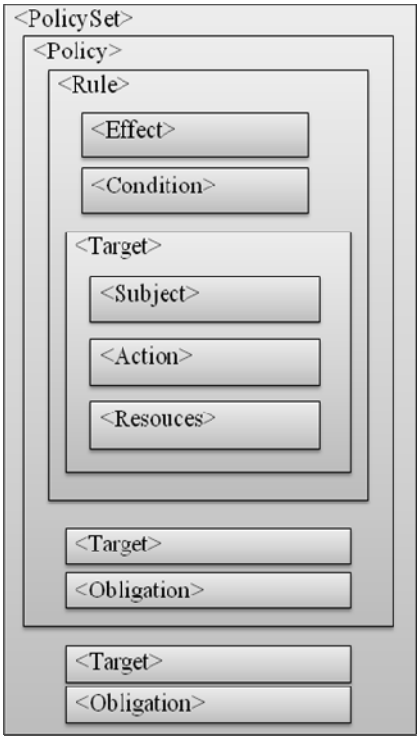

Fig 1: XACML Policy Language.

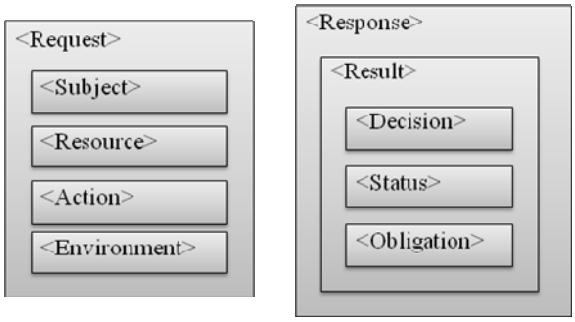

Fig. 2: XACML Request/Response syntax

The Effect indicates the rule-writer's intended consequence of a "TRUE" evaluation for the Rule. Two values are allowed: Permit and Deny.
Another element is the Obligation. It defines the requirements to be satisfied when allowing the requested Action. For example, a possible Obligation is to send an email to the administrator when the actual resource is accessed [14].
Additionally, the Policy contains a RuleCombiningAlgorithm field and the PolicySet contains PolicyCombiningAlgorithm field whose meaning is related to the decision process and will be explained in Section 2.4.

2.2 General syntax of XACML request and response

XACML also specifies the format used to convey an authorization request and related decision, i.e., the response. This format is called XACML Context; it is defined in a XML Schema (see Figure 2).
A Request Context consists of a set of attributes associated with the requesting subjects, the resource acted upon, the action being performed and the environment.
The Subject element specifies the entity making the access request. Resource element defines the resource to which the Subject has request access. Action element explains the action that the Subject wishes to perform on the resource (e.g., read, write or execute).
The Environment element describes the resource environment (e.g., date, time, etc.).
Subject, Resource, Action and Environment can contain multiple attribute values.
A Response Context contains one or more Results. They are obtained from the evaluation of the decision Request against the policy. The Decision can be one of the following strings: Permit, Deny, Not Applicable (if no applicable policies or rules could be found), or Indeterminate (if some error occurred during policy evaluation process). The Status returns optional information to characterize the error. Response may also include Obligations, if they are defined in the Policy or PolicySet evaluated.

2.3 The XACML high-level architecture

The XACML architecture includes four key components: the PEP (Policy Enforcement Point), the PDP (Policy Decision Point), the PAP (Policy Administration Point) and the PIP (Policy Information Point).
The PEP enforces access control by making decision requests and enforcing authorization decisions.
The PDP evaluates the applicable policy and renders an authorization decision.
The PAP creates security policies and stores them in an appropriate repository.
The PIP serves as the source of attributes or data required for policy evaluation. It manages all the information related to subject, resource and environment.
A simplified version of this model is depicted in Figure 3.
The typical work flow includes the following steps:
1. The PAP writes policies and policy sets and makes them available to the PDP.
2. The access requester sends an access request to the PEP. It may include attribute values of the subjects, resource, and environment.
3. The PEP constructs a standard XACML request Context and sends it to the PDP.
4. The PDP asks for any additional Subject, Resource, and Environment attribute values from the PIP.
5. The PIP obtains the requested attributes and returns them to the PDP.
6. The PDP asks to the PAP for the policies according to the request's target.
7. The PAP returns the request policies.
8. The PDP evaluates the related policy and returns the standard XACML Response Context to the PEP.
9. The PEP enforces the authorization decision.

2.4 Policy evaluation

According to the previous work flow, the PEP sends the authorization request to the PDP using the XACML Context. It should be composed almost exclusively of attribute values about Subject, Resource and Environment. The PEP is asking the PDP if a subject is allowed to

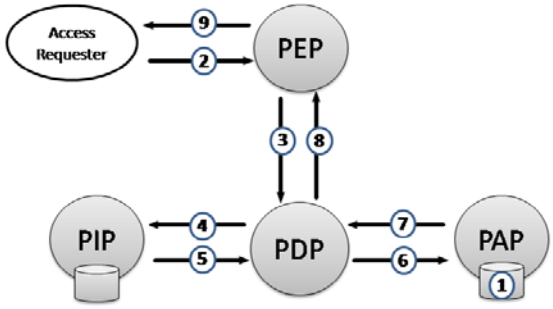

Fig. 3: XACML architecture and work flow

perform the specified action on the resource in the environment.
Then, when a request arrives to the PDP, it first locates all the Policies that matches the Target (by means of the PAP), after that it compares, using some particular functions, attribute values in the Request Context against attribute values contained in the policy.
When a policy is composed by a single Rule, the PDP take into account the Conditions defined in the actual Rule. If they evaluate to true, then the Rule's Effect (Permit or Deny) is returned to the PEP (in the Decision element of the Response Context). If the Condition evaluates to false then the PDP returns to the PEP the value Not Applicable (in the Decision element of the Response Context).
Otherwise if many Rule instances are contained in a Policy, PDP needs a method to reconcile the actions specified by all Rules. This is the meaning of Rule Combining Algorithm in the Policy element.
Analogously, for PolicySet, the Policy Combining Algorithm defines a procedure for arriving at an authorization decision given the individual results of evaluation of a set of policies.
In the XACML specification the standard combining algorithms defined are: Deny-overrides, Permit-overrides, First-applicable and Only-one-applicable.
For instance, the "Only-one-applicable" policy combining algorithm only applies to Policy elements. The result of this combining algorithm ensures that one and only one Policy or PolicySet is applicable.
If no Policy or PolicySet applies, then the result is "Not Applicable", but if more than one Policy or PolicySet is applicable, then the result is "Indeterminate" (see the XACML specification for further details).

3. XACML entities interaction

As we have seen the authorization process involves several entities (e.g., PDP, PEP, PAP and PIP) that must collaborate.
Although the XACML specification clearly describes components, it does not strictly define the interactions among these entities. This allows administrators or developers to adapt the system entities according to their requirements.
In the following sections we present more in details these entity interactions.

3.1 PDP-PEP

XACML does not define any mechanism for transmitting requests, responses, and attributes over a network.
It addresses only a configuration where the PDP and PEP are on the same system.
SAML (Security Assertion Markup Language), another OASIS standard [15], is traditionally used to this purpose. How to use SAML 2.0 to carry the XACML messages between the XACML actors is defined in the SAML profile for XACML [16].
According to the profile specifications there are two general elements used to manage the messages exchange: a Query (an extension of the SAML Request element) and a Statement (the response to the Query giving one or more results).
For PEP-PDP interaction the profile defines both a type of query (for Requests) and a type of statement (for Response). They are XACMLAuthzDecisionQuery and XACMLAuthzDecisionStatement.
XACMLAuthzDecisionQuery carries the XACML request that PEP sends to the PDP to request authorization decision.
XACMLAuthzDecisionStatement (an extension of the SAML Statement element) carries the XACML Response message sent from the PDP to the PEP.
When SAML is used, the PEP converts the XACML request into a XACMLAuthzDecisionQuery and sends it to the PDP. The PDP converts the SAML query into an XACML request and processes the request against the XACML policy. The XACML response is converted into a XACMLAuthzDecisionStatement and sent back to the PEP, which converts it back into an XACML response.
SAML does not provide message confidentiality only message integrity. If data being transmitted are sensitive, it must be protected using SSL/TLS or WS-Security.
Otherwise if the SAML protocol is being used without SSL/TLS, all SAML messages must be signed appropriately.

3.2 PDP-PAP

The XACML 2.0 Core Specification does not explicitly address how policies are made available to the PDP or controlled once they are available.
However, a XACML 2.0 entity, referred to as a PAP is functionally defined as "a system entity that creates a Policy or PolicySet". Additional references are contained within the XACML 2.0 Core Specification that explains the responsibilities of the PAP regarding such topics as composition of PolicySet and maintaining unique identifiers for Policy.
Two possible mechanisms for policy administration between a PAP and PDP are available: (1) a SAML-based request-response protocol and (2) a simple SAML Assertion-based storage format.
In the first case the PDP queries the PAP for policies using the mechanism described in [16]. This mechanism provides both a format to query a policy, and a format to carry the requested policy (an extension of SAML Statement). They are XACMLPolicyQuery and XACMLPolicyStatement.
XACMLPolicyQuery is used for requesting policies from the PAP. The element is extension of SAML Request element. For example, this query can be used to retrieve policies specific to a certain Target.

XACMLPolicyStatement carries the policies requested from the PAP.
In the second case a PAP may use a simple SAML Assertion-based storage format, for placing policies in a generic repository. It may be accessed directly by the PDP.

3.3 PDP-PIP

Attributes contained in a request is compared by PDP against attribute values in the appropriate policies (referred to the same request's Target).
If these attributes are not in the request, the PDP can ask them to the PIP.
In the XACML specifications are defined two mechanisms to retrieve attribute values: the Attribute Designator used to retrieve attribute values from a request (e.g., by specifying the name, type, and issuer of attributes) and the Attribute Selector that allow the PDP to look for attribute values through an XPath [17] query.
Using the Attribute Designator the PDP looks for that attribute value in the request, otherwise it use other source (e.g., LDAP directory) to retrieve the needed informations. This element contains a URN that identifies the attribute.
There are four kinds of Attribute Designator, one for each type of attributes in a request: Subject, Resource, Action, and Environment.
Additionally attributes can be divided into different categories. For example to support the notion of multiple subjects making a request, subject attributes are categorized (e.g., the user, the user's workstation, the user's network, etc.). Then Subject Attribute Designators can also specify a category to look in.
Otherwise the Attribute Selectors, through an XPath expression, can be used to resolve some set of attribute values in the request Context or in other location (e.g., a XML database).
Since the Attribute Designator and the Attribute Selector can return multiple values, XACML provides a special attribute type called a Bag.
Bags are unsorted value collections that allow duplicates, and are exclusively used by the PDP. It manages Bags by means of a set of ad hoc functions. If no matches are made, an empty bag is returned.
As an alternative to the XACML native methods, SAML Query and Statement extensions can also be used. They are AttributeQuery and AttributeStatement.
AttributeQuery (a standard SAML Request) may be used by the PIP to request attributes from Attribute Repositories (e.g. LDAP, etc). For example, this query can be used to retrieve administrator's email address.
AttributeStatement is the response to the attribute query that can contains one or more values.

4. XACML in action

In order to use XACML for access control in a web service scenario, all the described entities (PAP, PIP, PDP and PEP) must be available. The following subsections present available open source implementation and related issues.

4.1 Policy Administration Point

According to the XACML data-flow model, the first step to be accomplished is policy creation and storage by the PAP.
Several tools have been developed to implement this functionality: UMU XACML policy editor [18]. XACML-Studio (XS) [19]. Additionally, eXist, an open source native XML database, supports the creation of XACML policy [20] by means of a GUI.
Once the policy has been written and stored, it is necessary to verify its correctness. To this purpose, policy verification tool are available. These tools are used to formally check general properties of access control policies.
Examples of these tools are the ones developed by Hughes and Bultan [21] e by Fisler et al. [22].
Hughes and Bultan translate XACML policies to the Alloy language [23] and check their properties using the Alloy Analyzer.
Fisler et al. developed an XACML policy verification tool called Margrave [24] that verifies user-specified properties and performs change-impact analysis.
Additionally Zhang et al. [25] developed a model-checking algorithm with a supporting tool to evaluate access control policies written in RW languages, which can be converted to XACML.
These approaches support only a subset of the XACML policy specification language because it is challenging to generalize these verification approaches to support full featured XACML policies with complex conditions.
Some of these approaches also require the user to specify a set of properties in some formal language to be verified; however, these formally specified properties often do not exist in practice.
To avoid this problem E. Martin et al. [26] propose an approach for conducting conformance checking of XACML access control policies synthesizing first of all the concrete and desirable properties (from the policy under checking) and then feed the synthesized concrete properties to a policy verification tool or policy testing tools available.
At the same time many researchers are working on the automatic generation of XACML policies from business process specification such as BPMN [27] and WS-CDL [28].
Wolter et al. [29] define a mapping between the XACML and the BPMN meta-models [30] to provide a model-driven extraction of security policies from a business process model expressed in BPMN. The translation

process is done by means an XSLT converter that transforms modelled security constraints into XACML policies. Clearly correctness and completeness of authorization constraints must be verified into BPMN.
In [31] Robinson et al. describe the idea to automatically derive the minimal authorizations required for the collaboration starting from WS-CDL specification. Additionally they aims at enabling and disabling authorization rules on in a just in time manner that matches the control flow. Nevertheless, they proved the effectiveness of their approach only in a well-defined collaborative scenario.
Actually no "real" tools are available to automatically create XACML policies starting from business processes model, although some rough prototypes are available. Therefore despite the attempts to ease XACML policy creation, if you want to write your access control policies you'd better use a XACML policy editor, store them into a database and check them with a policy verification tool (e.g., the Margrave tool).

4.2 Policy Decision Point

The core of an access control architecture is the decision engine, called the PDP. We have therefore analysed the features and the performance of several available PDP implementations.
Sun Microsystems Laboratories has provided an open source implementation of the XACML written in Java and available from SourceForge [32]. The most important feature is a set of API to manage the PDP lifecycle. The API supports basic PDP operations for parsing policies and request/response documents, for deciding about applicability of policies and for evaluating requests against policies. Additionally, the API provides methods to manage standard attribute types, functions and combination algorithms, and to add new functionalities (e.g., new mechanisms for retrieving policies from PAP and attributes from PIP).
Based on this API, under the Apache License a PAP and a PDP have been implemented as web services that use Axis2. The project is called XACMLight [33]. It works theoretically in any J2EE compliant container but it was tested only on a native Axis2 server.
Other XACML implementations are XACML.NET 0.7 [34], Parthenon XACML policy engine [35], Enterprise-java-XACML [36], Herasaf [37] and XEngine[38].
We have exclude from our analysis Herasaf and XEngine because they are not available for testing.
Our opinion is that it is not possible to highlight a generic "perfect" choice among the available implementation, because user requirements must be taken into account.
The Parthenon Policy engine is the best choice if strict compliance to standard is a requirement, but since it is not an open source software, we exclude it from our classification.
The Sun PDP fails in supporting just 3% of the XACML mandatory functionalities compared to XACML.NET implementation that fails in supporting 9% of the same ones [39].
Nevertheless the Sun implementation presents some shortcomings leading to non optimal performance:
1) it does not have any cache mechanism for policy or evaluation result and this slows down the policy evaluation process;
2) it simply matches any new request against each available policy and for this reason it is not appropriate if the number of policies increases.
Starting from these considerations, it is clear that the two important stages in an XACML policy evaluation for performance analysis are:

1. loading of policy/policies from disk to main memory;
2. evaluation of request against the loaded policies.

Turkmen et al. [40] created an experimental schema to test performance of any generic PDP implementation. To stay in close contact with the real world usage patterns for XACML, they created an experimental schema with different elements representing various usage scenarios. Each element provides a different view to the access control problem on diverse environments. Their policy test suite is:

- large number of policies (over 1000);
- large number of rules (over 1000 in a single policy);
- 10 policies that have some similar rule inside.

According to the authors' results, Enterprise-java-XACML is the best choice in terms of policy evaluation time because it has many mechanisms for efficient policy evaluation such as target indexing and policy and result caching. In particular target indexing significantly speeds up the policy applicability search process for the given request. However it was the worst in policy loading.
Otherwise XACMLight inherits from Sun's implementation the problem in managing large number of policies.
According to the authors' results, XEngine is also an efficient policy evaluation engine. XEngine first converts a textual XACML policy to a numerical policy. Then it converts a numerical policy with complex structures to a numerical policy with a normalized structure. Finally, it converts the normalized numerical policy to tree data structures for efficient processing of requests. The experimental results show that XEngine is more efficient than the Sun PDP, and the performance difference grows almost linearly with the number of rules.

4.3 Policy Information Point

The PIP acts as source of attributes and in a web service scenario holds information about how to associate HTTP request attributes and sender information to XACML subject, resource and environment identifiers according to the XACML schema.

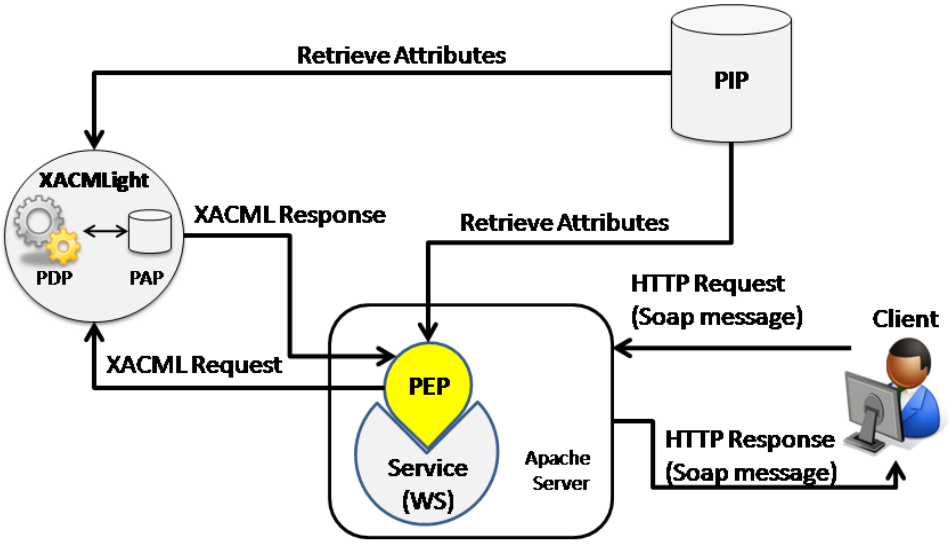

Fig. 4: PEP integrated in WS code

PIP can be built by using the Sun implementation via the com.sun.xacml.finder package that provides to the PDP support for retrieving attributes not already present in the request. It aims to resolve resource identifiers or to generate real-time values.

4.4 Policy Enforcement Point

The PEP is the entity that enforces PDP's decisions. The main role of this agent is to be the interface between the managed application and the PDP. Hence, it translates the requests which are expressed in the application specific language into the standardized protocol language understood by the PDP. The PEP can also get additional information (attributes of subjects, resources and environments) from the PIP that will help the PDP to take its decision.
Finally it translates the PDP's decisions into the application specific language and enforces the decision.
When using XACML in a Web service or Web application, PEP can be:

1. embedded in the source code (Figure 5);
2. placed as an additional module between the client and the web service container (e.g., Apache, JBoss) (Figure 6).

In the first case the web service must be able to generate request and process answers according the XACML syntax for request/response as depicted in figure 5.
Already most of the code that you need for building a PEP is provided by Sun's implementation in the com.sun.xacml.ctx package (which represents the context schema). More details are available in the SUN's Developer Guide.
Otherwise, it is possible to use a module that intercepts the SOAP messages direct to and from the web service.
The module can maintain the XACML Context and other information (e.g., the state).
Laborde et al. [41] explain this idea. Their work is inspired by the Apache HTTP server. Apache provides a basic core HTTP server that can be enhanced by additional and configurable modules. In the same way, they propose a core PEP that calls additional modules to translate requests, get further information in different locations and translate/enforce PDP's decisions.
As an alternative the module can be implemented by means of a JAX-WS APIs as SOAP handler that implements the PEP functionality [42].
Actually application independent PEPs are not available because it must consider web service related attributes.
For usability, ease of maintainability and extensibility, we suggest to implement a PEP as separated module acting as SOAP message interceptor that can be integrate in a HTTP server.

5. Actual PEP-PDP usage

5.1 Interoperability

XACML is an industry accepted standard that provides a well defined structure to create rules and policy sets to make complex authorization decisions.
The actual interoperability between XACML entities from different vendors was demonstrated in two different times.
The first-ever XACML interoperability demonstration was hosted at the Burton Group Catalyst Conference in June 2007. The demonstration event occurred with eight vendors showcasing the results of their work.

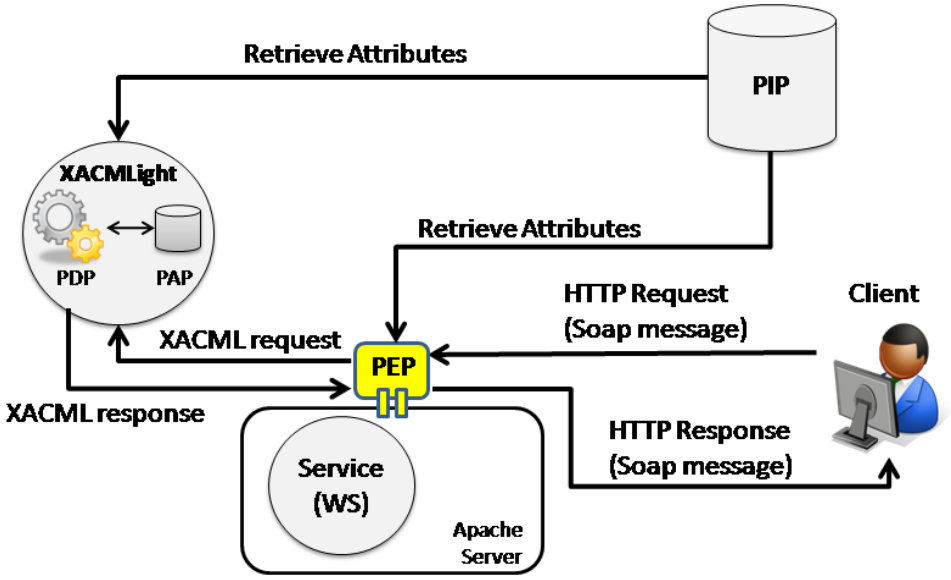

Fig. 5: PEP as additional module

There were two particular use cases in this demo, which required interoperability between vendor implementations of PEP, PDP and PAP: (1) Authorization Decision and (2) Policy Exchange.

Here we are interested in first case where four different scenarios were defined: (1) Customer Access, (2) Customer Transaction, (3) Account Manager Access and (4) Account Manager Approval.

The Authorization Decision Interoperability Demo aimed to demonstrating that XACML version 2.0 requests generated by the PEP of Vendor A (PEP-A) are properly evaluated by the PDP of Vendor B (PDP-B), where Vendor A and Vendor B may be any of the vendors participating in the Interoperability Demo .

At the RSA Conference 2008 in San Francisco, nine organizations came together to demonstrate, to the second time, interoperability simulating a real world scenario provided by the U.S Department of Veterans Affairs.

Vendors showed how XACML obligations can provide capabilities in the policy decision making process. The use of XACML obligations with SAML was also highlighted.

In looking at the two Interoperability Demo scenario documents [43] [44], it is clear that some specific choices were made to make the Demo work:

1) Use of the SAML 2.0 Profile for XACML 2.0 which defines a Request/Response mechanism for Request and Response Context.

2) Implementation of the XACML Interface of the PDP as a SOAP Interface which accepts a XACML authorisation decision query (by means of SAML syntax) and returns a XACML authorisation decision statement.

5.2 Shortcomings in XACML

XACML is just an access control policy language, thus it cannot be considered as a basis for a full authorization infrastructure. Even in the version 2 of the standard there are some deficiencies that must know and worked around:

- It does not define how user credentials are validated (the specifications only talks about user attributes).
- It does not specifies how policies are securely stored and retrieved;
- It does not foresee integration with auditing mechanisms. In fact if XACML is used, it is hard to keep track of access rights of users or monitoring what a user can do in the system.
- It does not say how environmental attributes are securely obtained;
- It does not specify what an obligation is and how manage it.

Additionally, current version of the standard does not cover the concept of delegation and specifically the delegation of policies, both in static and in dynamic way. A delegate is defined as “A person authorized to act as representative for another; a deputy or an agent”. Delegation of authority is the act of one user with a privilege giving it to another user (a delegate), in accordance with some delegation policy.

The delegation concept is used to cater for the temporary transfer of access rights. However the ability of a user to delegate (or revoke) access rights to another must be tightly controlled by security policies. This requirement is critical in systems allowing cascaded delegation of access rights. A delegation policy permits subjects to grant privileges, which they possess (due to an existing authorization policy), to grantees to perform an action on

their behalf (e.g., passing read rights to a printer spooler in order to print a file). This last issue is directly addressed by the forthcoming XACML version 3.

5.3 Open Issues.

XACML provides a standard, flexible and fine-grained mechanism for defining and enforcing access controls across distributed systems but according to our opinion it is not mature yet. Still, many issues remain to be resolved before the emergence and adoption of XACML as stable standard. We propose and present a list of the open issue to improve XACML:

1. standardizing the attribute values;
2. standardizing the obligations that are returned, along with a protocol for talking to an Obligations Service;
3. integration with other authorization frameworks that use different policy languages (e.g. EPAL [45], Ponder, PERMIS, P3P [46]).
4. retrieving attributes from multiple sources;
5. support for Sticky Policies[1] [47];
6. building application independent PEPs and obligation services.

6. Conclusion

XACML provides a standard mechanism to satisfy interoperability requirements among the entities of an access control system. However, it does not address all the concepts that could be necessary to manage distributed access control system (e.g., delegation). Most likely it will be improved in its next version but for current use these shortcomings should be addressed in other ways.
Several implementations of the basic XACML components are available. But our analysis showed that they are not equivalent in terms of functionality and performance. Therefore a careful selection should be performed before using them in a practical implementation. Despite these issues we conclude that XACML is ready for prime time access-control systems and will likely be the basis for several future improvements in this area.

References

[1] "Bit Torrent", www.bittorrent.com.

[2] "Grid Computing", http://en.wikipedia.org/wiki/Grid_computing .

[3] "Mobile Agent", http://en.wikipedia.org/wiki/Mobile_agent

[4] M. Thompson, A. Essiari, S. Mudumbai, "Certificate-based Authorization Policy in a PKI Environment," ACM Trans. on Information and System Security, August 2003

[5] N. Damianou, N. Dulay, E. Lupu, M. Sloman, "The Ponder Policy Specification Language". Proc. Policy 2001, Workshop on Policies for Distributed Systems and Networks, Bristol, 29-31 Jan. 2001, Springer-Verlag LNCS 1995, pp. 18-39

[6] "The Web Services Policy Framework (WS-Policy)", www.w3.org/Submission/WS-Policy/

[7] "PERMIS", www.permis.org

[8] "eXtensible Access Control Markup Language (XACML)" Version 2.0, OASIS Standard, 1 February 2005, http://docs.oasisopen.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[9] "Web Services Architecture", http://www.w3.org/TR/ws-arch

[10] "XACML profile for RBAC", http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf

[11] "ITU Recommendation X.812", http://www.itu.int/rec/T-REC-X.812/

[12] "ISO/IEC 10181-7:1996", www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18200

[13] RFC-2573 "A Framework for Policy-based Admission Control

[14] C. Bettini, S. Jajodia, X. S. Wang, D. Wijesekera, Provisions and obligations in policy rule management and security applications, Proc. 28th VLDB Conference, Hong Kong, China, 2002

[15] "Security Assertion Markup Language (SAML)", OASIS Standard, March 2005, www.oasis-open.org/committees/security/

[16] "SAML profile for the XACML", OASIS February 2005, docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-saml-profile-spec-os.pdf

[17] "XML Path Language (XPath)", W3C, November 1999, http://www.w3.org/TR/xpath [rif]

[18] "UMU-XACML-Editor", Version 1.2.0, University of Murcia (UMU), Spain , http://xacml.dif.um.es

[19] "XACML-Studio (XS)", http://xacml-studio.sourceforge.net

[20] "Open Source Native XML Database", http://exist.sourceforge.net

[21] G. Hughes, T. Bultan, "Automated verification of access control policies". Technical Report 2004-22, Dept. of Computer Science, University of California, Santa Barbara, 2004

[22] K. Fisler, S. Krishnamurthi, L. Meyerovich, M. Tschantz, "Verification of change-impact analysis of access-control policies". In International Conference on Software Engineering, pp. 196–205, 2005

[23] D. Jackson, I. Shlyakhter, M. Sridharan, "A micromodularity mechanism". In Proc. 8th ESEC/FSE, pages 62–73, 2001

[24] "An API for XACML Policy Verification and Change Analysis", Margrave, Brown University, Providence (RI), http://www.cs.brown.edu/research/plt/software/margrave/

[25] N. Zhang, M. Ryan, D. P. Guelev, "Synthesising verified access control systems in XACML". In Proc. 2004 ACM

[1] Sticky policies are strictly associated to users' data and drive access control decisions and privacy enforcement.

workshop on Formal Methods in Security Engineering, pages 56–65, 2004

[26] E. Martin, V. C. Hu, J. Hwang, T. Xie, "Conformance Checking of Access Control Policies Specified in XACML", Proceedings of the 1st IEEE International Workshop on Security in Software Engineering, July 23-27, 2007

[27] "Business Process Modeling Notation (BPMN)", http://www.bpmn.org

[28] "Web Services Choreography Description Language (WS-CDL)", W3C, November 2005, www.w3.org/TR/ws-cdl-10/

[29] C. Wolter, A. Schaad, C. Meinel "Deriving XACML Policies from Business Process Models", WISE 2007 Workshops, LNCS 4832, pp. 142–153, 2007

[30] C. Wolter, A. Schaad, "Modeling of Authorization Constraints in BPMN". BPM 2007. Proc. of the 5th Int. Conf. on Business Process Management (2007)

[31] P. Robinson, F. Kerschbaum, A. Schaad, "From business process choreography to authorization policies", 20th IFIP WG 11.3 Working Conference on Data and Applications Security (2006)

[32] Sun's XACML implementation, Version 1.3, July 2004, sunxacml.sourceforge.net/guide.html

[33] "XACMLight", http://sourceforge.net/projects/xacmllight

[34] "XACML.NET", http://mvpos.sourceforge.net/

[35] "Parthenon XACML policy engine", http://www.parthcomp.com/xacml_toolkit.html

[36] "Enterprise java XACML", http://code.google.com/p/enterprise-java-xacml/

[37] "HERASAF", http://herasaf.org/xacmlimpl/index.html

[38] F. C. Alex, X. Liu, "Xengine, "A fast and scalable XACML policy evaluation engine," Department of Computer Science, Michigan State University, East Lansing, Michigan, Tech. Rep. MSU-CSE-08-2, March 2008.

[39] N. Li, J. Hwang,T. Xie, "Multiple-Implementation Testing for XACML Implementations," in 11th Workshop on Testing, Analysis and Verification of Web Software, 2008.

[40] F. Turkmen, B. Crispo "Performance Evaluation of XACML PDP Implementations", SWS'08, October 31, 2008, Fairfax (VA, USA)

[41] R. Laborde, M. Kamel, F. Barrere, A. Benzekri, "PEP=Point to Enhance Particularly", IRIT, UPS, Toulouse, POLICY 2008, 2-4 June 2008 , New York (NY, USA)

[42] "JAX-WS APIs as SOAP handler that implement PEP functionality", https://prof.hti.bfh.ch/fileadmin/home/due1/app_sec/Integrating_XACML_into_JAX-WS_and_WSIT-20081127.pdf

[43] Burton Group Conference 2007 Interoperability Demo scenario document, http://xml.coverpages.org/XACMLv20-Interop-Burton2007InteropDemo-document.zip.

[44] RSA 2008 Interoperability Demo scenario document, http://xml.coverpages.org/XACMLv20-Interop-RSA2008-28011.zip

[45] "Enterprise Privacy Authorisation Language (EPAL)", http://xml.coverpages.org/epal.html.

[46] "Platform for Privacy Preferences Project (P3P)", http://www.w3.org/P3P/

[47] M. Mont, S. Pearson, P. Bramhall, "Towards Accountable Management of Identity and Privacy: Sticky Policies and Enforceable Tracing Services". TrustBus 2003 workshop, 2003

Piervito Scaglioso received his M.Sc. degree in 2006. He is a PhD student in Computer Engineering at the Politecnico di Torino. His research interests are in Policy-based Systems and their application to manage Access Control systems. Additionally he is interested in Wireless Sensor Network and he works on the creation of a novel security protocol for access control to wireless sensor data.

Cataldo Basile holds a M.Sc. and a Ph.D. in Computer Engineering from the Politecnico di Torino where is currently a research assistant. His research is concerned with policy-based management of security in networked environments, automatic refinement of policies to device configurations and general models for detection, resolution and reconciliation of specification conflicts.

Antonio Lioy holds a M.Sc. in Electronic Engineering *summa cum laude*, and a Ph.D. in Computer Engineering, both from Politecnico di Torino, Italy. He is currently Full Professor of Computer Engineering at the the Politecnico di Torino and leads the TORSEC security group. His research interests are in the area of policy-based security, network security and PKI. Prof. Lioy is a member of the PSG (Permanent Stakeholders' Group) of ENISA (the European Network and Information Security Agency).