XML-based Access Control Languages

C.A. Ardagna E. Damiani S. De Capitani di Vimercati P. Samarati
Dipartimento di Tecnologie dell'Informazione
Università degli Studi di Milano
26013 Crema, Italy
{ardagna,damiani,decapita,samarati}@dti.unimi.it

Abstract

One of the most challenging problems in managing large, distributed, and heterogeneous networked systems is specifying and enforcing security policies regulating interactions between parties and access to services and resources. Recent proposals for specifying and exchanging access control policies adopt XML-based languages. XML appears in fact a natural choice as the basis for the common security-policy language, due to the ease with which its syntax and semantics can be extended and the widespread support that it enjoys from all the main platform and tool vendors.

In this chapter, we first investigate the basic concepts behind access control design and enforcement, and point out different security requirements that may need to be taken into consideration in designing an access control language for Internet information systems. We then focus on XML-based access control languages and, in particular, on the eXtensible Access Control Markup Language (XACML), a recent OASIS standardization effort. XACML is designed to express authorization policies in XML against objects that are themselves identified in XML. The language can represent the functionalities of most policy representation mechanisms.

1 Introduction

Accessing information on the global Internet has become an essential requirement of the modern economy. Information is however one of, if not the most valuable asset of an organization. An important requirement of any system is then to protect its *data* and *services* against unauthorized disclosure (*secrecy* or *confidentiality*) and unauthorized or improper modifications (*integrity*), while at the same time ensuring their availability to legitimate users (*no denial-of-service* or *availability*) [10, 13, 18]. The problem of ensuring protection has existed since information has been managed. However, as technology advances and information management systems become more and more powerful, the problem of enforcing information security also becomes more critical. A fundamental component in enforcing protection is represented by the *access control* service whose task is to control every request to data/services maintained by a system and determining whether the request should be granted or denied. The access control service establishes the kinds of regulations (policies) that can be stated, through an appropriate specification language, and then enforced by the access control mechanism enforcing the service. By using the provided interface, security administrators can specify the access control policy (or policies) that should be obeyed in controlling access to the managed resources.

The definition of access control policies to be fed into the access control system is far from being a trivial process. One of the major difficulties lies in the interpretation of, often complex and sometimes ambiguous, real world security policies and in their translation in well defined unambiguous rules enforceable by the computer system. Many real world situations have complex policies, where access decisions depend on the application of different rules coming, for example, from laws practices, and organizational regulations. A security policy must capture all the different regulations to be enforced and, in addition, must consider all possible additional threats due to the use of computer systems. Given the complexity of the scenario, it is therefore important that the access control language be expressive and flexible enough to accommodate all the different requirements that may need to be expressed, while at the same time be simple both in terms of use (so that specifications can be kept under control) and implementation (so to allow for its verification). Some of the main concepts/features that an access control language expressing security policies should include are discussed below [11]:

- *Interchangeable policy format.* Protection requirements on the data need to be defined by using a format both human- and machine- readable, easy to inspect and interchange. This format should be simple to complement and check for being compliant with externally defined regulations; also, it should be simple enough to be readily understood by non-specialists.

- *Support for fine- and coarse-specifications.* The access control system should allow rules to be referred to specific accesses, providing fine-grained reference to the subjects and objects in the system. However, fine-grained specifications should be supported, but not forced. In fact, requiring the specification of access rules with reference to every single user and object in the system would make the administration task a heavy burden. Beside, groups of users and collections of objects often share the same access control requirements. The access control system should then provide support for authorizations specified for groups of users, groups of objects, and possibly even groups of actions [15]. Also, in many organizational scenarios, access needs may be naturally associated with *organizational activities*; the access control system should then support authorizations referred to organizational roles [19].

- *Conditional authorizations.* Protection requirements may need to depend on the evaluation of some conditions [18]. Conditions can be in the simple form of system's predicates, such as the date or the location of an access (e.g., 'Employee can access the system *from 9 am to 5 pm*'). Conditions can also make access dependent on the information being accessed (e.g., 'Managers can read payroll data of *the employees they manage*').

- *Policy combination and conflict-resolution.* If multiple modules (e.g., for different authorities or different domains) exist for the specification of access control rules, the access control system should provide a means for users to specify how the different modules should interact, for example, if their union (maximum privilege) or their intersection (minimum privilege) should be considered. Also, when both permissions and denials can be specified, the problem naturally arises of how to deal with *incompleteness*, that is, existence of accesses for which no rule is specified, and *inconsistency*, that is, the existence of accesses for which both a denial and a permission are specified. Dealing with incompleteness–requiring the authorizations to be complete would be very impractical–requires support of a *default* policy either imposed by

the system or specified by the users. Dealing with inconsistencies require support for *conflict resolution* policies. While, among the different conflict resolution policies that can be thought of (see [18] for a deeper treatment), some solutions may appear more natural than others, none of them represents "the perfect solution". Whichever approach we take, we will always find one situation for which the approach does not fit. Therefore any conflict resolution policy imposed by the access control mechanism itself will always result limiting. On the other side, support of negative authorizations does not come for free, and there is a price to pay in terms of authorization management and less clarity of the specifications. However, the complications brought by negative authorizations are not due to negative authorizations themselves, but to the different semantics that the presence of permissions and denials can have, that is, to the complexity of the different real world scenarios and requirements that may need to be captured. There is therefore a trade-off between expressiveness and simplicity. Consequently, current systems try to keep it simple by adopting negative authorizations for exception support, imposing specific conflict resolution policies, or supporting a limited form of conflict resolution.

Recently, several proposals have been introduced for access control to distributed heterogeneous resources from multiple sources based on the use of attribute certificates [5]. These proposals are often based on the use of logic languages, which are not immediately suited to the Internet context, where simplicity and easy integration with existing technology must be ensured. XML-based access control languages seems more suitable for this context and are also well suited for the interchange of policies. Two relevant access control languages using XML are WS-Policy [8] and XACML [17]. Based on the WS-Security [4], WS-Policy provides a grammar for expressing Web service policies. The WS-Policy includes a set of general messaging related assertions defined in WS-PolicyAssertions [6] and a set of security policy assertions related to supporting the WS-Security specification defined in WS-SecurityPolicy [22]. In addition, WS-PolicyAttachment [7] defines how to attach these policies to Web services or other subjects such as service locators. The eXtensible Access Control Markup Language (XACML) [17] is a language for the expression of authorization policies in XML against objects that are themselves identified in XML. While XACML and WS-Policy share some common characteristics, XACML has the advantage of enjoying an underlying policy model as a basis, resulting in a clean and unambiguous semantics of the language. For this reason, in this chapter we illustrate XACML as our choice of language.

The remainder of this chapter is structured as follows. Section 2 introduces the basic features of XACML. Section 3 describes the XACML policy language model and presents an example of XACML policy. Finally, Section 4 gives our conclusions.

2 XACML: basic characteristics and data flow model

The eXtensible Access Control Markup Language (XACML) [17] is the result of a recent OASIS standardization effort proposing an XML-based language to express and interchange access control policies. XACML is designed to express authorization policies in XML against objects that are themselves identified in XML. The language can represent the functionalities of most policy representation mechanisms and has standard *extension points* for defining new functions, data types,

combining logic, and so on. We now describe the main features of XACML and then shows the data-flow model.

2.1 XACML features

The major functionalities offered by XACML can be summarized as follows.

- *Combination policy support.* XACML provides a method for combining policies independently specified. Different entities can then define their policies on the same resource. When an access request on that resource is submitted, the system has to take into consideration all these policies. XACML defines three elements for the specification of access control policies: `Rule`, `Policy`, and `PolicySet`. The `Rule` element corresponds to the traditional concept of *authorization*: it defines who can access to what resource and under which conditions. The `Policy` element consists of a set of rules and specifies how combining the results of their evaluation. Finally, the `PolicySet` element contains a set of `Policy` or `PolicySet`.

- *Combining algorithms support.* Since both a `Policy` and `PolicySet` element can contain multiple policies or rules, each of which can evaluate to different access control decisions, XACML needs to define a method for reconciling such decisions. XACML supports different combining algorithms, each representing a way of combining multiple decisions into a single decision. To this purpose, XACML defines two attributes, namely `RuleCombiningAlgId`, and `PolicyCombiningAlgId`. The first attribute indicates a method for combining the individuals results of evaluation of a set of rules. The second attribute indicates a method for combining the individuals results of evaluation of a set of policies.

- *Attribute support.* XACML supports the definition of policies based on properties (attributes) associated with subjects and resources other than their identities. This allows the definition of powerful policies based on generic properties associated with subjects (e.g., name, address, occupation) and resources. To this purpose, XACML provides two elements, namely `SubjectAttributeDesignator` and `ResourceAttributeDesignator`, that together with `SubjectMatch` and `ResourceMatch` elements allow to identify a particular subject and resource attribute, respectively.

- *Operators support.* XACML includes some built-in operators for comparing attribute values and provides a method of adding non-standard functions.

- *Multiple subjects.* XACML allows the definition of more than one subject relevant to a decision request.

- *Policy distribution support.* Policies can be defined by different parties and enforced at different enforcement points. Also, XACML allows one policy to contain or refer to another.

- *Implementation independency.* XACML provides an abstraction-layer that isolates the policy-writer from the implementation details. This means that different implementations should operate in a consistent way, regardless of the implementation itself. As we will see later on, XACML defines a canonical form for the request and response, called *XACML context*.

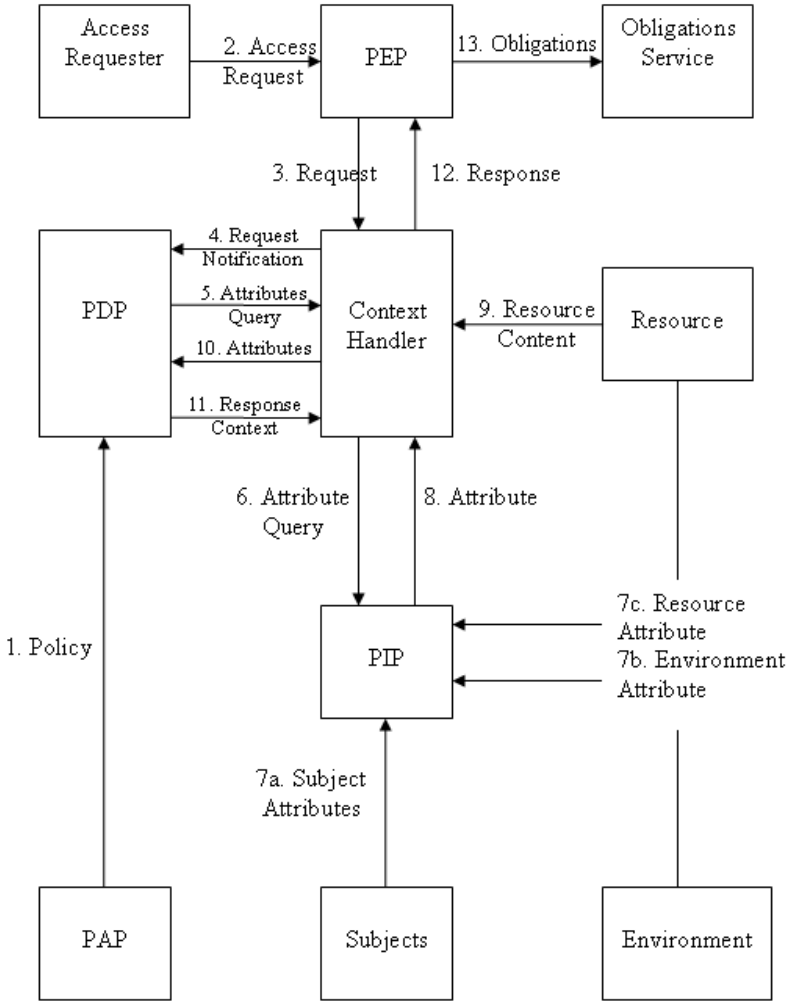

Figure 1: XACML overview [17]

- *Obligations support.* XACML provides a method for specifying some action, called *obligations*, that must be fulfill in conjunction with the policy enforcement. The element that provides this feature is the `Obligation` element.

2.2 Data-flow model

The main entities involved in the XACML domain are illustrated in Figure 1. The standard gives a definition of these concepts that we summarize as follows.

- The *Policy Evaluation Point* (PEP) module enforces the access decision taken by the decision point.

- The *Policy Decision Point* (PDP) module receives an access request and interacts with the PAP that encapsulates the information needed to identify the applicable policies. It then evaluates the request against the applicable policies and returns the authorization decision to the PEP module.
- The *Policy Administration Point* (PAP) module retrieves the policies applicable to a given access request and returns them to the PDP module.
- The *Policy Information Point* (PIP) module provides attributes values about the subject, resource, and action (the function to be performed).
- The *Context Handler* translates the access requests in a native format into a canonical format.
- The *Environment* provides a set of attributes that are relevant to take an authorization decision and are independent of a particular subject, resource, and action.

A typical scenario for using XACML is when someone wants to take some action on a resource. For instance, suppose a physician wants to access a patient's record for inquiry only. The physician would log on to the hospital information system, enter the patient identifier, and retrieve the corresponding record. In this case, the hospital application would make a request to the PEP module that protects the patient inquiry function. The PEP would then create a request based on the physician's attributes, the resource requested, and any other information related to the request. The PEP module then sends this request to the PDP which evaluates the request. To this purpose, the PDP interacts with the PAP which retrieves the policies applicable to the request and returns them to the PDP module. More precisely, data flows through an XACML model by the following steps (see Figure 1):

- The requester sends an access request to the PEP module.
- The PEP module sends the access request to the context handler which translates the original request in an *XACML request context* by inquiring the PIP to obtain attributes of the subject, resource, action, and environment. The PIP retrieves the requested attributes and returns them to the context handler. Optionally, the context handler includes the resource in the context.
- The context handler sends the XACML request to the PDP. The PDP identifies the applicable policies by means of the PAP module and retrieves the required attributes and, possibly, the resource from the context handler. The PDP then evaluates the policies and returns the *XACML response context* to the context handler.
- The context handler translates the XACML response context to the native format of the PEP and returns it to the PEP together with an optional set of obligations.
- The PEP fulfills the obligations and, if the access is permitted, performs the access. Otherwise, the PEP denies access.

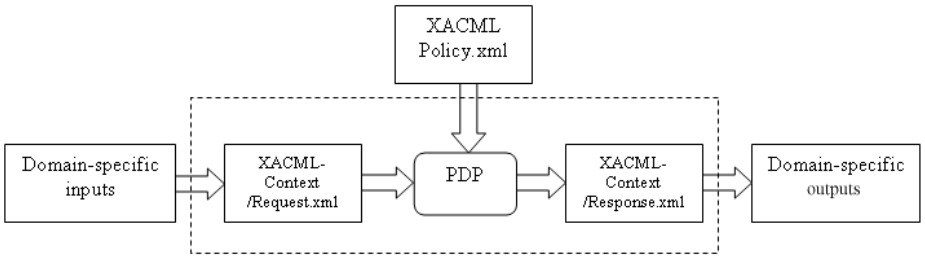

Figure 2: XACML context [17]

As described above, the XACML Context defines a canonical form of the request/response managed by the PDP. Any implementation has to translate the attribute representations in the application environment (e.g., SAML, .NET, Corba) in the XACML context (see Figure 2). As an example, consider SAML [20] that is the most successful standard protocol handling authentication information across transactions between parties. SAML uses tagged sets of user attributes to represent subject-related information encapsulated inside service requests. An application can then provide a SAML message that includes a set of attributes characterizing the subject making the access request. This message has to be converted to the XACML canonical form and, analogously, the XACML decision has then to be converted to the native format.

3 Policy Language Model

The main conceptual difference between XACML and other XML-based access control languages is that XACML relies on a model that provides a *formal* representation of the access control security policy and its working. This modeling phase is essential to ensure a clear and unambiguous language which could otherwise be subject to different interpretations and uses. This can be obviously a serious problem especially in the access control area, where access decisions have to be deterministic [1, 2]. Figure 3 illustrates the *XACML policy language model*. The main concepts of interests are *rule*, *policy*, and *policy set*.

3.1 PolicySet, Policy and Rule

An XACML policy has as root element either a `Policy` or a `PolicySet`. A `PolicySet` is a collection of `Policy` or `PolicySet`. An XACML policy consists of a set of *rules*, a *target*, an optional set of *obligations*, and a *rule combining algorithm*. A `Rule` specifies a permission (`permit`) or a denial (`deny`) for a subject to perform an action on an object. A `Target` basically consists of a simplified set of conditions for the subject, resource, and action that must be satisfied for a policy to apply to a given request. If all the conditions of a `Target` are satisfied, then its associated `Policy` (or `Policyset`) applies to the request. If a policy applies to all entities of a given type, that is all subjects, actions, or resources, an empty element, named `AnySubject`, `AnyAction`, `AnyResource`, respectively, is used. An `Obligation` is an operation that has to be performed in conjunction with the enforcement of an authorization decision. For instance, an obligation can state that all accesses

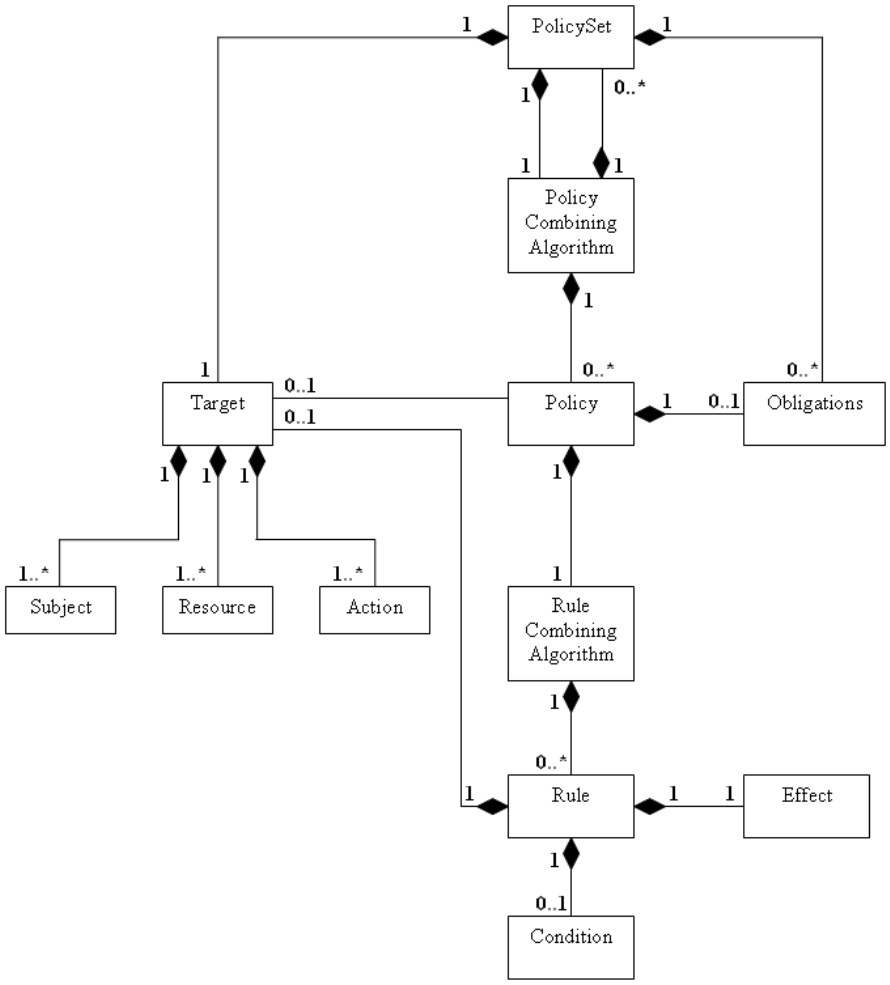

Figure 3: XACML policy language model [17]

on medical data has to be logged. Obligations are passed back in the response from the PDP to the PEP. Note that only policies which are evaluated and have returned a response of `permit` or `deny` can return obligations. More precisely, an obligation is returned only if the effect of the policy matches the value specified in the `FullfillOn` attribute associated with the obligation. This means that if a policy evaluates to `indeterminate` or `not applicable`, then the associated obligations are not returned to the PEP. Each `Policy` also defines a rule combining algorithm used for reconciling the decisions each rule makes. The final decision value, called *authorization decision*, inserted in the XACML context by the PDP is the value of the policy as defined by the rule combining algorithm. XACML defined different combining algorithms. Examples of these are the following.

- *Deny overrides.* If there exists a rule that evaluates to `deny` or, if all rules evaluates to `not applicable`, then the result is `deny`. If all rules evaluates to `permit`, then the result is

`permit`. If some rules evaluate to `permit` and some evaluate to `not applicable`, then the result is `permit`.

- *Permit overrides.* If there exists a rule that evaluates to `permit`, then the result is `permit`. If all rules evaluate to `not applicable`, then the result is `deny`. If some rules evaluate to `deny` and some evaluate to `not applicable`, then the result is `deny`.

- *First applicable.* Each rule is evaluated in the order in which it appears in the `Policy`. For each rule, if the target matches and the conditions evaluate to true, then the result is the effect (`permit` or `deny`) of such a rule. Otherwise, the next rule is considered.

- *Only-one-applicable.* If more than one rule applies, then the result is `indeterminate`. If no rule applies, then the result is `not applicable`. If only one policy applies, the result coincides with the result of evaluating that rule.

In summary, according to the selected combining algorithm, the authorization decision returned to the PEP can be `permit`, `deny`, `not applicable` (when no applicable policies or rules could be found), or `indeterminate` (when some errors occurred during the access control process).

The `PolicySet` element consists of a set of *policies*, a *target*, an optional set of *obligations*, and a *policy combining algorithm*. The policy, target, and obligation components are as described above. The policy combining algorithms define how the results of evaluating the policies in the policy set has to be combined when evaluating the policy set. This value is then inserted in the XACML response context by the PDP.

As said before, a rule specifies the actual conditions under which access is to be allowed or denied. The components of a rule are a *target*, an *effect*, and a *condition*. The target defines the set of resources, subjects, and actions to which the rule is intended to apply. The effect of the rule can be `permit` or `deny`. The condition represents a boolean expression that may further refine the applicability of the rule. Note that the `target` element is an optional element: a rule with no target applies to all possible requests.

An important feature of XACML is that a rule is based on the definition of attributes corresponding to specific characteristics of a subject, resource, action, or environment. For instance, a physician at an hospital may have the attribute of being a researcher, a specialist in some field, or many other job roles. According to these attributes, that physician can be able to perform different functions within the hospital. As another example, a particular function may be dependent on the time of the day (e.g., access to the patient records can be limited to the working hours of 8:00 AM to 6:00 PM). When an access request is sent from the PEP to the PDP, that request is mainly composed of attributes that will be compared to attribute values in a policy to make an access decision. Attributes are identified by the `SubjectAttributeDesignator`, `ResourceAttributeDesignator`, `ActionAttributeDesignator`, and `EnvironmentAttributeDesignator` elements. These elements use the `AttributeValue` element to define the value of a particular attribute. Alternatively, the `AttributeSelector` element can be used to specify where to retrieve a particular attribute. Note that both the attribute designator and attribute selector elements can return multiple values. To this reason, XACML provides an attribute type called *bag*. A bag is an unordered collection and can contain duplicates values for a particular attribute. In addition, XACML defines other standard value types such as string, boolean, integer, time, and so on. Together with these attribute types,

XACML also defines operations to be performed on the different types such as equality operation, comparison operation, string manipulation, and so on.

3.2 XACML request and response

XACML defines a standard format for expressing requests and responses. More precisely, the original request submitted by the PEP is translated through the context handler in a canonical form then forwarded to the PDP to be evaluated. Such a request contains attributes for the subject, resource, action, and, optionally, for the environment. Each request includes exactly one set of attributes for the resource and action and at most one set of environment attributes. There may be multiple sets of subject attributes each of which is identified by a category URI. Figure 4 illustrates the XSD Schema of the request.

```
<xs:element name="Request" type="xacml-context:RequestType">
<xs:complexType name="RequestType">
  <xs:sequence>
    <xs:element ref="xacml-context:Subject" maxOccurs="unbounded">
    <xs:element ref="xacml-context:Resource">
    <xs:element ref="xacml-context:Action">
    <xs:element ref="xacml-context:Environment" minOccurs="0">
  </xs:sequence>
</xs:complexType>
```

Figure 4: XACML request schema

A response element contains one or more results each of which correspond to the result of an evaluation. Each result contains three elements, namely `Decision`, `Status`, and `Obligations`. The `Decision` element specifies the authorization decision (i.e., `permit`, `deny`, `indeterminate`, `not applicable`), the `Status` element indicates if some error occurred during the evaluation process, and the optional `Obligations` element states the obligations that the PEP must fulfill. Figure 5 illustrates the XSD Schema of the response.

```
<xs:element name="Result" type="xacml-context:ResultType">
<xs:complexType name="ResultType">
  <xs:sequence>
    <xs:element ref="xacml-context:Decision">
    <xs:element ref="xacml-context:Status">
    <xs:element ref="xacml-context:Obligations" minOccurs="0">
  </xs:sequence>
  <xs:attribute name="ResourceId" type="xs:string" use="optional">
</xs:complexType>
```

Figure 5: XACML response schema

```
<Policy PolicyId="Policy1" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides">
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
            http://www.example.com/forum/private.html
          </AttributeValue>
          <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#anyURI"
          AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
  <Rule RuleId="ReadRule" Effect="Permit">
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              read
            </AttributeValue>
            <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="group"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        moderator
      </AttributeValue>
    </Condition>
  </Rule>
</Policy>
```

Figure 6: XACML policy

3.3 An XACML example

We illustrate a simple example of XACML policy, request and response. Suppose that there is a corporation named FORUM CORP that defines a high level policy as follows:

Any member of the `moderator` group can read the web page
`www.example.com/forum/private.html`

Figure 6 shows the XACML policy corresponding to the high level policy of FORUM CORP. The policy applies to requests on the `http://www.example.com/forum/private.html` resource. It has one rule with a target that requires an action of `read` and a condition that applies only if the subject is a member of the group `moderator`. Suppose now that a user belonging to group `moderator` and with email `user1@example.com` wants to read the `www.example.com/forum/private.html` web page. The corresponding XACML request is illustrated in Figure 7(a). This request is compared with the previous XACML policy. The result is that the user is allowed to access the requested web page. The corresponding XACML response is illustrated in Figure 7(b).

4 Conclusions

In this chapter, we have discussed the basic concepts of access control and illustrated the main features of the eXtensible Access Control Markup Language (XACML). XACML is a powerful and expressive language with many benefits [21]: it allows the unification of access control languages; policies do not have to be rewritten in different languages; developers do not have to invent new policy languages and write code to support them; it encourages reusability; multi-application tools for managing and writing access control policies will be unified; it allows extensions to the access control language to accommodate other access control policies; it allows one policy to contain or refer to another.

References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. An XPath-based preference language for P3P. In *Proc. of the World Wide Web Conference*, Budapest, Hungary, May 2003.

[2] C. Ardagna and S. De Capitani di Vimercati. A comparison of modeling strategies in defining xml-based access control languages. *CSSE*, 2004.

[3] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (epal 1.1). IBM Research Report. http://www.zurich.ibm.com/security/enterprise-privacy/epal.

[4] B. Atkinson and G. Della-Libera et al. Web services security (WS-Security). `http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-security.asp%`, April 2002.

[5] P. Bonatti and P. Samarati. A unified framework for regulating access and information release on the web. *Journal of Computer Security*, 10(3):241–272, 2002.

```
<Request>
  <Subject>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
    DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
      <AttributeValue>user1@example.com</AttributeValue>
    </Attribute>
    <Attribute AttributeId="group" DataType="http://www.w3.org/2001/XMLSchema#string"
    Issuer="administrator@example.com">
      <AttributeValue>moderator</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      <AttributeValue>http://www.example.com/forum/private.html</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
    DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>read</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

(a)

```
<Response>
  <Result>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

(b)

Figure 7: An example of XACML request (a) and response (b)

[6] D. Box et al. Web services policy assertions language (WS-PolicyAssertions) version 1.1. `http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policyassert%ions.asp`, May 2003.

[7] D. Box et al. Web Services Policy Attachment (WS-PolicyAttachment) version 1.1. `http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policyattach%ment.asp`, May 2003.

[8] D. Box et al. Web services policy framework (WS-Policy) version 1.1. http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policy.asp, May 2003.

[9] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Securing SOAP E-services. *International Journal of Information Security (IJIS)*, 1(2):100–115, February 2002.

[10] E. Damiani, S. De Capitani di Vimercati, and P. Samarati. Towards security XML web services. In *Proc. of the 2002 ACM Workshop on XML Security, Washington, DC, USA*, November 2002.

[11] S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Access control: Principles and solutions. *Software – Practice and –¿ – Experience*, 33(5):397–421, April 2003.

[12] S. Feldman. The Changing Face of E-Commerce. *IEEE Internet Computing*, 4(3):82–84, May/June 2000.

[13] B. Galbraith, W. Hankinson, A. Hiotis, M. Janakiraman, D. Prasad, and R. Trived. *Professional Web Services Security*. Wrox Press Ltd., December 2002.

[14] J. Hine, W. Yao, J. Bacon, and K. Moody. An architecture for distributed OASIS services. In *Proc. of the IFIP/ACM International Conference on Distributed Systems Platforms nd Open Distributed Processing*, Hudson River Valley, New York, USA, April 2000.

[15] S. Jajodia, P. Samarati, M. Sapino, and V. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(2):18–28, June 2001.

[16] H. Koshutanski and F. Massacci. An access control framework for business processes for web services. In *Proc. of the 2003 ACM Workshop on XML security*, Fairfax, Virginia, November 2003.

[17] OASIS eXtensible Access Control Markup Language (XACML) version 1.1. `http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specific%ation-1.1.pdf`.

[18] P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, LNCS 2171. Springer-Verlag, 2001.

[19] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[20] Security assertion markup language (SAML) v1.0. http://www.oasis-open.org/committees/download.php/3400/oasis-sstc-saml-1.1-pdf-xsd.zip.

[21] SUN. *Introduction to XACML*, June 2003. http://sunxacml.sourceforge.net.

[22] Web services security policy (WS-SecurityPolicy), December 2002. `http://www-106.ibm.com/developerworks/library/ws-secpol/`.