CHAPTER 4
EXTENSIBLE ACCESS CONTROL MARKUP LANGUAGE

XACML is an XML-based language for access control that has been developed by OASIS (Rissanen 2013). XACML defines both an access control policy language, and a request and response language.

The policy language is an ABAC mechanism, used to construct expressions that make up an access control policy, which specifies the security mechanisms. In other words, the policy language defines the required constraints and conditions to a subject for accessing a resource, and carries out an action through a specific environment. The policy language is an extensible, flexible, highly expressive, standards-based, and general-purpose language. Moreover, it enables the specification of fine-grained policies, used to access the control to resources.

The request and response language describes the subjects making requests for accessing resources, and renders the authorization decisions granting or denying the access request. XACML is used not only for controlling the access to XML documents, but also for controlling the access to any type of resources.

XACML defines four logical modules:

1. Policy Administration Point (PAP) creates policies or policy sets, and manages, and stores them in an appropriate repository.
2. Policy Decision Point (PDP) evaluates an applicable policy and conveys an authorization decision.

3. Policy Enforcement Point (PEP) carries on an access control by conveying the decision requests to a PDP, and enforcing the authorization decisions returned by the PDP.

4. Policy Information Point (PIP) acts as the source of the required attribute values of a subject, a resource, an action, and an environment for a policy's evaluation.

The main functions offered by XACML can be summarized as follows (Ardagna et al 2007, Ardagna et al 2009):

1. **Policy combination**: XACML provides a method for combining policies specified independently. Therefore, different entities can define their policies on the same resource. Hence, when an access request on that resource is submitted, the system takes into account all the applicable policies.

2. **Combining algorithms**: XACML provides different combining algorithms; each one provides a way of reconciling multiple decisions into a single decision.

3. **Attribute-based restrictions**: XACML provides the definition of policies based on the attributes of subjects (e.g., name and address) and resources (e.g., creation date and type). Moreover, it includes built-in operators for comparing the attribute values and provides a method for adding nonstandard functions.
4. **Policy distribution**: Policies can be defined by different parties and carried out at different enforcement points. Moreover, XACML allows one policy to contain another one or refers to it.

5. **Implementation independence**: XACML provides an abstraction layer that insulates the policy-writer from its implementation details. This insulation guarantees that different implementations are executed consistently.

6. **Obligations**: XACML provides a method for specifying actions, called obligations, which must be executed in conjunction with the applicable policies that have decisions.

The main concepts of all XACML policies are a `<PolicySet>`, `<Policy>`, and `<Rule>`, which represent a single access control policy. The root element of each XACML policy document is exactly one `<PolicySet>`, `<Policy>`, or `<Rule>`. The `<Policy>` consists of one or more `<Rule>` elements. A `<policy>` has at least one `<Rule>`. The `<Rule>` includes the core logic of an XACML policy. The decision logic of the rules is included in a `<Condition>`. The `<Condition>` is a Boolean function that refines the applicability of the rule. If the `<Condition>` returns true, then the rule's Effect (Permit or Deny) is returned. If the `<Condition>` returns false, the PDP returns to the PEP the value NotApplicable.

If many `<Policy>` elements are contained in a `<PolicySet>`, the PDP needs a way to reconcile the effects returned by all policies. Hence, the concept of the Policy Combining Algorithm is introduced in a `<PolicySet>`. The final decision value of the policy-combining algorithm is called the authorization decision. Similarly, if many `<Rule>` elements are contained in a `<Policy>`, the PDP needs a way to reconcile the effects returned by all rules.
Hence, the concept of the Rule Combining Algorithm is introduced in each <Policy>.

Moreover, XACML provides a feature, called a <Target>, which is a set of simplified attribute values for a subject, a resource, an action, and an environment that must be satisfied for a <PolicySet>, <Policy>, or <Rule> to be applicable to a given request. If all the attribute values of a <Target> are satisfied with the associated <PolicySet>, <Policy>, or <Rule>, then the associated <PolicySet>, <Policy>, or <Rule> applies to the request (Scaglioso et al 2008, Hsieh et al 2010).

4.1 XACML POLICY LANGUAGE

XACML policies are XML documents with one <PolicySet>, one <Policy>, or one <Rule> as the root element. A <PolicySet> contains zero or more <PolicySet> elements (optionally), zero or more <Policy> elements (optionally), one <Target> (required), one identifier for the policy-combining algorithm (required), zero or one <ObligationExpressions> (optionally), and zero or one <AdviceExpressions> (optionally). A <Policy> contains one or more <Rule> elements (at least one <Rule>), one identifier for the rule-combining algorithm, one <Target>, zero or one <ObligationExpressions>, and zero or one <AdviceExpressions>. A <Rule> contains zero or one <Target>, zero or one <Condition>, one Effect attribute, zero or one <ObligationExpressions>, and zero or one <AdviceExpressions>. A <Target> is a set of attribute values to identify uniquely a subject, a resource, an action, and an environment that must be satisfied for a <PolicySet>, <Policy>, or <Rule> to be applicable to a given request. If all the conditions of a <Target> are satisfied with the associated <PolicySet>, <Policy>, or <Rule>, then the associated <PolicySet>, <Policy>, or <Rule> applies to the request. The <Target> must appear as a child of a <PolicySet> and <Policy>. However, it

4.1.1 Rule

A <Rule> contains a RuleId attribute as an identifier, zero or one <Description>, zero or one <Target>, zero or one <Condition>, one Effect attribute, zero or one <ObligationExpressions>, and zero or one <AdviceExpressions>. The required RuleId attribute is a string that assigns a unique name to the <Rule>. The optional <Description> contains a free description to the rule.

The optional <Target> defines the set of attribute values of the requests to which the rule is proposed to apply in the form of a logical expression on the attributes of the request. If the matches defined by the target are satisfied by the attributes of the request, the rule is applicable to the request. The rule is proposed to be applied to all entities of a particular data type, if this entity is omitted from the <Target>. If the <Target> is omitted from a <Rule>, the target of the <Rule> will be the same as the <Target> of its parent <Policy>.

The required Effect attribute of the rule indicates the rule consequence of a True evaluation for the rule. The Effect attribute value is Permit or Deny.

The optional <Condition> is a Boolean expression that must be satisfied (be true) for the rule to be assigned its Effect attribute value. It refines the applicability of the rule. For example, in the sentence "Only allow logins from 10 am to 6 pm", the condition indicates that, the access is allowed only in the interval [10 am - 6 pm]. If the <Condition> is omitted or evaluates to true, the condition value will be True. The condition value will be False if
the <Condition> evaluates to false. The condition value will be Indeterminate, if an operational error occurred during the evaluation, such as missing attributes, network errors while retrieving rules, division by zero, or syntax errors in the decision request or in the rule. Therefore, the <Rule> is evaluated as follows:

1. (If the <Rule> has not a <Target>, or the <Target> matches the attributes of the request) and the <Condition> evaluates to true, the rule value will be the value of the Effect attribute (Permit or Deny).

2. (If the <Rule> has not a <Target>, or the <Target> matches the attributes of the request) and the <Condition> evaluates to false, the rule value will be NotApplicable.

3. (If the <Rule> has not a <Target>, or the <Target> matches the attributes of the request) and the <Condition> evaluates to Indeterminate, the rule value will be Indeterminate {P}, if the Effect attribute value is Permit, or Indeterminate {D}, if the Effect attribute value is Deny. Indeterminate {P} means an Indeterminate from a policy or rule, which will be evaluated to Permit but not Deny. Indeterminate {D} means an Indeterminate from a policy or rule, which will be evaluated to Deny but not Permit.

4. If the <Target> of the <Rule> does not match the attributes of the request, the rule value will be NotApplicable. The Rule's condition value will not be considered.

5. If the <Target> of the <Rule> matching evaluates to Indeterminate, the rule's value will be Indeterminate {P}, if the Effect attribute value is Permit, or Indeterminate {D}, if
the Effect attribute value is Deny. The rule's condition value will not be considered.

4.1.1.1 Expression

The <Condition> contains one <Expression>. The <Expression> must return a value of type "http://www.w3.org/2001/XMLSchema#boolean". The <Expression> may be an <AttributeDesignator>, an <AttributeSelector>, an <Apply>, an <AttributeValue>, or a <Function>.

The <AttributeDesignator> is used as a pointer to identify a bag of all the values of the named attributes in the request subject by its category, identifier, and data type. A named attribute is the matched attribute of the request subject in the values of Category, AttributeId, and DataType attributes. Therefore, the <AttributeDesignator> may be used to retrieve one or more attribute values from the <Attributes> of the request subject context.

The <AttributeSelector> is used to retrieve a bag of unnamed and uncategorized attribute values from the <Content> of the request resource context. The values are retrieved from the nodes selected by applying an XPath expression to the XML content of the <Content> of the request resource context. Since both the <AttributeDesignator> and <AttributeSelector> may return multiple values, XACML provides an attribute type, called bag, which is an unordered collection that can contain duplicate values for a particular attribute.

The <Apply> represents a function and its arguments. The <Apply> contains a required FunctionId identifier that determines the function that will be applied to the arguments of the <Apply>, zero or more <Expression> elements that represent the arguments of the function, and zero or one <Description>, which is a free description to the <Apply>.
The <Function> is used to assign a name to a function, which is an argument to another function defined by its parent <Apply>. It contains a required FunctionId identifier.

The <AttributeValue> includes a literal attribute value, and it has a required DataType attribute that represents the data type of the literal attribute value.

### 4.1.1.2 Obligation Expressions

An obligation specifies an action described by the attribute values and must be executed, when the result of the rule is Permit/Deny. Only rules that are evaluated and have an effect of permit or deny, can return obligations. For example, an obligation states that, "all accesses to an employee's data have to be logged". Another obligation states, "send an email to the administrator when the actual resource is accessed". The <Rule> specifies the obligation, using an optional <ObligationExpressions>. This element contains one or more <ObligationExpression> elements.

The <ObligationExpression> contains a required ObligationId attribute, FulfillOn attribute, and zero or more <AttributeAssignmentExpression> elements.

The required ObligationId attribute is an obligation identifier. The required FulfillOn attribute determines for which effect (Permit/Deny) this obligation must be carried out by the PEP.

The <AttributeAssignmentExpression> retrieves the attribute values that describe the action specified by the obligation. Each retrieved attribute value will be included in an <AttributeAssignment> of an <Obligation> within the response. If the evaluation of the
<AttributeAssignmentExpression> retrieves an empty bag of values, there is no corresponding <AttributeAssignment> in the <Obligation> of the response. The <AttributeAssignmentExpression> contains a required AttributeId attribute, an optional Category attribute, and one <Expression>.

The AttributeId is the value of the AttributeId attribute for the retrieved attribute values included in the corresponding <AttributeAssignment> of the <Obligation> within the response.

The Category attribute indicates the category of the retrieved attribute values, which may be a subject category or a resource category. Hence, the category attribute is used to distinguish among the attribute values retrieved from the request context. The value of the Category attribute in the <AttributeAssignmentExpression> must be equal to the value of the Category attribute in the corresponding <AttributeAssignment>.

The required <Expression> retrieves the required attribute values for an obligation using an <AttributeDesignator>, <AttributeSelector>, or <AttributeValue>. Therefore, when a PDP evaluates a rule, which has an effect of permit or deny, and the effect matches the value of the FulfillOn attribute contained in an <ObligationExpression>, the PDP evaluates the expressions in the <AttributeAssignmentExpression>, includes the retrieved attribute values in an <Obligation>, and returns an <Obligations>, which is containing the <Obligation>, within the response context to the PEP.

4.1.1.3 Advice Expressions

In addition to the obligation, the <Rule> may contain an advice that specifies an action described by the attribute values, and must be executed when the result of the rule is Permit/Deny. Only rules that are evaluated and have an effect of permit or deny can return advices. The <Rule> specifies the
advice using an optional <AdviceExpressions>. This element contains one or more <AdviceExpression> elements.

The <AdviceExpression> contains a required AdviceId attribute, AppliesTo attribute, and zero or more <AttributeAssignmentExpression> elements.

The required AdviceId attribute is an advice identifier. The required AppliesTo attribute determines for which effect (Permit/Deny) this advice must be carried out by the PEP.

The <AttributeAssignmentExpression> retrieves the attribute values that describe the action specified by the advice. Each retrieved attribute value will be included in an <AttributeAssignment> of an <Advice> within the response. If the evaluation of the <AttributeAssignmentExpression> retrieves an empty bag of values, there is no corresponding <AttributeAssignment> in the <Advice> of the response. The <AttributeAssignmentExpression> contains a required AttributeId attribute, an optional Category attribute, and one <Expression>.

The AttributeId is the value of the AttributeId attribute for the retrieved attribute values included in the corresponding <AttributeAssignment> of the <Advice> within the response.

The Category attribute indicates the category of the retrieved attribute values, which may be a subject category or a resource category. Hence, the category attribute is used to distinguish among the attribute values retrieved from the request context. The value of the Category attribute in the <AttributeAssignmentExpression> must be equal to the value of the Category attribute in the corresponding <AttributeAssignment>. 
The required <Expression> retrieves the required attribute values for the advice using an <AttributeDesignator>, <AttributeSelector>, or <AttributeValue>. Therefore, when a PDP evaluates a rule that has an effect of permit or deny, and the effect matches the value of the AppliesTo attribute contained in an <AdviceExpression>, the PDP evaluates the expressions in the <AttributeAssignmentExpression>, includes the retrieved attribute values in an <Advice>, and returns an <AssociatedAdvice>, which is containing the <Advice>, within the response context to the PEP. Contrary to the obligation, an advice may be safely ignored by the PEP (Rissanen 2013).

4.1.2 Policy

A <Policy> contains a PolicyId attribute as an identifier, zero or one <Description>, one or more <Rule> elements, an identifier for the rule-combining algorithm, one <Target>, zero or one <ObligationExpressions>, and zero or one <AdviceExpressions>. The required PolicyId attribute assigns a unique name to the <Policy>. The optional <Description> contains a free description to the policy.

The policy may contain an optional <ObligationExpressions> and <AdviceExpressions> that are explained in the subsection 4.1.1. Only policies that are evaluated and have a decision of permit or deny, can return obligations or advices. When a PDP evaluates a policy that has an authorization decision (permit or deny), and the decision matches the value of the FulfillOn or AppliesTo attribute contained in an <ObligationExpression> or <AdviceExpression> respectively, it evaluates the expressions in the <AttributeAssignmentExpression>, includes the retrieved attribute values in an <Obligation> or <Advice>, and returns an <Obligations> containing the <Obligation>, or an <AssociatedAdvice> containing the <Advice>, within the response context to the PEP (Rissanen 2013).
4.1.2.1 Target

The writer of a <Policy> or <PolicySet> may define the required <Target> of the <Policy> or <PolicySet>, or the <Target> may be calculated from the <Target> of its inner components <PolicySet>, <Policy>, or <Rule>. If the policy's writer defines the <Target> of a <Policy>, any <Rule>, contained in the <Policy> and has the same <Target> as its parent <Policy>, may omit its <Target>. These <Rule> elements inherit the <Target> of their parent <Policy>. If the <Target> needs to be calculated, there are two logical ways that can be used to calculate the <Target> of the <Policy> or <PolicySet>.

In the first method, the <Target> of the root <Policy> or <PolicySet> is calculated as the union of all the <Target> elements of the inner components (children of the root) <PolicySet>, <Policy>, or <Rule>. In this case, the <Target> of the root will be applicable to any decision request that matches the <Target> of at least one child element.

In the second method, the <Target> of the root <Policy> or <PolicySet> is calculated as the intersection of all the <Target> elements of the inner components <PolicySet>, <Policy>, or <Rule>. In this case, the <Target> of the root will be applicable only to a decision request that matches the <Target> of every child element of the root.

4.1.2.2 Rule Combining Algorithms

The required rule-combining algorithm identifier explains how the results of the rules' evaluations are reconciled to evaluate the policy; i.e., it specifies the value of the authorization decision (policy's value) that is placed in the response context by the PDP.
XACML defines different combining algorithms, namely, Deny overrides algorithm, Permit overrides algorithm, First applicable algorithm, and Only-one-applicable algorithm. The Deny overrides algorithm states that:

1. If there is a rule that evaluates to Deny, the policy evaluates to Deny.
2. If all the rules evaluate to NotApplicable, the policy evaluates to NotApplicable.
3. If all the rules evaluate to Permit, the policy evaluates to Permit.
4. If some rules evaluate to Permit and some evaluate to NotApplicable, the policy evaluates to Permit.

The Permit overrides algorithm states that:

1. If there is a rule that evaluates to Permit, the policy evaluates to Permit.
2. If all the rules evaluate to NotApplicable, the policy evaluates to NotApplicable.
3. If all the rules evaluate to Deny, the policy evaluates to Deny.
4. If some rules evaluate to Deny and some evaluate to NotApplicable, the policy evaluates to Deny.

The First applicable algorithm states that:

1. The rules are evaluated in the listing order, in which they appear in the <Policy>. 

2. For each rule, if the rule's target matches the request's target and the condition evaluates to true, the policy evaluates to the rule's Effect attribute value (Permit or Deny), for the selected rule.

3. If the target evaluates to false or the condition evaluates to false, evaluate the next rule in the order.

4. If there is no rule in the order, the policy evaluates to NotApplicable.

5. If an error occurs during the evaluation of the target or condition of a rule, stop the rules' evaluation, and the policy evaluates to Indeterminate.

The Only-one-applicable algorithm states that:

1. If there is more than one rule, which are applicable by their target, the policy evaluates to Indeterminate.

2. If there is no applicable rule by its target, the policy evaluates to NotApplicable.

3. If only one rule is applicable by its target, the policy result is the same as the evaluation result of that rule.

4. If an error occurs during the evaluation of a rule, the policy evaluates to Indeterminate.

According to the selected combining algorithm, the authorization decision returned to the PEP can be Permit, Deny, NotApplicable (when no applicable rules could be found), or Indeterminate (when some errors occurred during the access control process) (Ardagna 2007, Rissanen 2013).
4.1.3 Policy Set

A <PolicySet> contains a PolicySetId attribute as an identifier, zero or one <Description>, zero or more <PolicySet> elements, zero or more <Policy> elements, one <Target>, an identifier for the policy-combining algorithm, zero or one <ObligationExpressions>, and zero or one <AdviceExpressions>.

The required PolicySetId attribute assigns a unique name to the <PolicySet>. The optional <Description> contains a free description to the policy set. The <Policy> and <Target> are presented above in the subsection 4.1.2. The required policy-combining algorithm identifier explains how the results of the policies' evaluations are reconciled together to evaluate the policy set; i.e., it specifies the decision value (policy set's value) placed in the response context by the PDP. The policy-combining algorithms are similar to the rule-combining algorithms that are discussed above.

Only policy sets that are evaluated and have a decision of permit or deny, can return obligations or advices. When a PDP evaluates a policy set that has an authorization decision (permit or deny), and the decision matches the value of the FulfillOn or AppliesTo attribute contained in an <ObligationExpression> or <AdviceExpression> respectively, it evaluates the expressions in the <AttributeAssignmentExpression>, includes the retrieved attribute values in an <Obligation> or <Advice>, and returns an <Obligations> containing the <Obligation>, or an <AssociatedAdvice> containing the <Advice>, within the response context to the PEP. An example of an XACML policy set document is depicted in Figure 4.1:
<PolicySet PolicySetId="pls-0001" PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides">
  <Description>
    It is the description of the policy set
  </Description>
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            Omar
          </AttributeValue>
          <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id">
            <DataType "http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectAttributeDesignator>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <Policy PolicyId="pol-0001" RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides">
    <Description>
      It is the description of the policy
    </Description>
    <Rule RuleId="rul-0001" Effect="Permit"/>
  </Policy>
</PolicySet>

Figure 4.1 Example of an XACML Policy Set Document
In Figure 4.1, the required attribute PolicySetId assigns a name to the <PolicySet>. The name of a <PolicySet>, <Policy>, or <Rule> has to be unique for a given PDP, so that there is no ambiguity if a <PolicySet>, <Policy>, or <Rule> is referenced from another <PolicySet>, <Policy>, or <Rule>.

The required identifier PolicyCombiningAlgId specifies the algorithm used to reconcile the results of the various <Policy> elements that may be in the <PolicySet>. The Deny-overrides policy-combining algorithm states that, if there is a policy that evaluates to Deny, the policy set evaluates to Deny. If all the policies evaluate to NotApplicable, the policy set evaluates to NotApplicable. If all the policies evaluate to Permit, the policy set evaluates to Permit. If some policies evaluate to Permit and some evaluate to NotApplicable, the policy set evaluates to Permit.

The optional <Description> has a text, which is a description of the <PolicySet>. The policy set's <Target> indicates that this policy set is applicable to any request from a subject, whose name is Omar, to execute any action on any resource. The attribute values in a given request are compared with the attribute values in the <Target>. If all the attribute values are matched together, the request is applicable. Hence, the request is further checked against the <Target> of the inner <Policy> and the <Target> of the inner <Rule> elements. If the attribute values of the request and those in the <Target> do not match, the request is not applicable. If an error occurred during the evaluation, such as missing attributes, network errors while retrieving policies, division by zero during a policy evaluation, or syntax errors in the decision request or in the policy, the request is indeterminate. If the <Target> is empty, the <PolicySet> is applicable to any access request.

The <Match> matches the attribute values in the <Attributes> of the request context with the embedded attribute value of the policy set's
<Target> in the <AttributeValue>. The required MatchId attribute specifies the matching function that will be used to compare the attribute values of the request and those of the <Target>. The value of this attribute must be of the type xs:anyURI. The required <AttributeValue> specifies the attribute data type, and the attribute value of the subject. The <AttributeDesignator> is used as a pointer to identify a bag of all the values of the named attributes in the request by its category, identifier, and data type. The <Match> must contain either the <AttributeDesignator> or <AttributeSelector> once, which is called the Required Choice; i.e., only one of them will appear once in the <Match>.

Moreover, in Figure 4.1, the required attribute PolicyId assigns a unique name to the <Policy>. Each <Policy> must have a unique identifier for a given PDP. The required identifier RuleCombiningAlgId specifies the algorithm that will be used to reconcile the results of the various <Rule> elements of the <Policy>. The Deny-overrides rule-combining algorithm states that, if there is a rule that evaluates to Deny, the policy evaluates to Deny. If all the rules evaluate to NotApplicable, the policy evaluates to NotApplicable. If all the rules evaluate to Permit, the policy evaluates to Permit. If some rules evaluate to Permit and some evaluate to NotApplicable, the policy evaluates to Permit.

The policy's <Target> describes the decision requests to which this policy applies. If the attribute values in a decision request do not match the attribute values specified in the policy's <Target>, the remainder of the policy (the <Rule> elements of the <Policy>) does not need to be evaluated and the policy result is NotApplicable. If the <Target> is empty, the policy is applicable to any decision request.

The optional <Description> has a text, which is a description of the <Policy>. The <Policy> has only one <Rule>. The required attribute RuleId assigns a unique name to the <Rule>. Each <Rule> must have a unique
identifier for a given PDP. The required Effect attribute of the <Rule> specifies that, the effect of this rule will be Permit when the rule evaluates to True; i.e., the access request should be permitted. If the rule evaluates to False, the rule result will be NotApplicable. If an error occurs during the evaluation of the rule, the rule result will be Indeterminate.

The <Target> of a <Rule> describes the decision requests to which this rule applies. If the attribute values in a decision request do not match the attribute values specified in the rule's <Target>, the remainder of the rule (the <Condition>) does not need to be evaluated, and the rule's value will be NotApplicable. If the <Target> is empty, the rule is applicable to any decision request. Since the <Rule> in Figure 4.1 has not a <Target>, the <Rule> will inherit the empty <Target> of its parent <Policy>, and hence, it is applicable to any decision request. Since the <Rule> in Figure 4.1 has not a <Condition>, the rule's condition evaluates to True (Ardagna et al 2007, Xu et al 2011, and Rissanen 2013).

4.2 XACML REQUEST AND RESPONSE

In addition to the format of the policy language, XACML specifies the format of an authorization request and its response. This format is called the XACML Context. The Request Context is composed of a set of simplified attribute values for a <Subject>, a <Resource>, an <Action>, and an <Environment> that must be satisfied for a <Target> of a <PolicySet>, <Policy>, or <Rule>.

The <Subject> describes the person making the access request. The <Resource> describes the resource to which the subject has an access request. The <Action> specifies the actions that the subject needs to carry out on the resource. The <Environment> describes the resource environment using a set of attributes, which are relevant to take an authorization decision, and are
independent of a particular subject, resource, and action, such as date or time. For example, in the following access request, "Allow the head of a department of a faculty to update the information of the research scholars from the research scholars' web site on the faculty server". The subject is the head of the department, the target resource is the research scholars' web site on the faculty server, and the action is updating the information of the research scholars.

The <Request> contains one or more <Attributes> elements. Each <Subject>, <Resource>, <Action>, and <Environment> is associated with an <Attributes> to specify its attribute values. There may be multiple <Attributes> elements with different categories for each component. The required <Attributes> describes the attributes of the request context by using a set of <Attribute> elements associated with an attribute, called Category. The <Attributes> contains a Category attribute, an optional xml:id attribute, zero or more <Attribute> elements, and zero or one <Content>.

The Category attribute is a required attribute, which is used to specify the attribute category of the contained attributes. The Category attribute is used to distinguish among the attributes of a subject, a resource, an action, and an environment.

For a subject who needs the access request, the value of the category is "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject". For a subject who receives the results of the request, the value of the category is "urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject". For a resource, the value of the category is "urn:oasis:names:tc:xacml:3.0:attribute-category:resource". For an action, the value of the category is "urn:oasis:names:tc:xacml:3.0:attribute-category:action". For an environment, the value of the category is "urn:oasis:names:tc:xacml:3.0:attribute-category:environment".
The optional xml:id attribute assigns a unique identifier to the <Attributes>. The optional <Content> includes the contents of a resource in a free XML document format. For example, the <Content> may include a part of an XML document, and the <AttributeValue> may include an xpath for this part of the XML document. The <AttributeSelector> may refer to the <Content>.

The optional <Attribute> specifies the data type and attribute values of the attributes, which belong to the category of the request. It contains a required AttributeId attribute, and one or more <AttributeValue> elements. The required AttributeId attribute assigns a unique identifier to the <Attribute>. The <AttributeDesignator> may refer to the <Attribute> using the AttributeId attribute.

In a <Subject>, when the value of the AttributeId is "urn:oasis:names:tc:xacml:1.0:subject:subject-id", it indicates that, the <AttributeValue> will contain the name of the subject. In a <Resource>, when the value of the AttributeId is "urn:oasis:names:tc:xacml:1.0:resource:resource-id", it indicates that, the <AttributeValue> will contain the resource to which access is requested. In an <Action>, when the value of the AttributeId is "urn:oasis:names:tc:xacml:1.0:action:action-id", it indicates that, the <AttributeValue> will contain the action for which access is requested. In an <Environment>, when the value of the AttributeId is "urn:oasis:names:tc:xacml:1.0:environment:current-dateTime", it indicates that, the <AttributeValue> will contain date and time.

The required <AttributeValue> includes the value of the attribute that belongs to the category of the request, and has a required attribute, called DataType. The content of the <AttributeValue> may be empty, occur once, or multiple times. An example of a request context is shown in Figure 4.2:
Figure 4.2 (Continued)
<AttributeValue
   DataType = "http://www.w3.org/2001/XMLSchema#string">
   sign
</AttributeValue>
</Attribute>
</Attributes>
</Action>
</Environment>
<Attributes Category = "urn:oasis:names:tc:xacml:3.0:attribute-category:environment">
  <Attribute
     AttributeId = "urn:oasis:names:tc:xacml:1.0:environment:current-dateTime">
    <AttributeValue
       DataType = "http://www.w3.org/2001/XMLSchema#dateTime">
      2010-06-14T11:12:03
    </AttributeValue>
  </Attribute>
</Attributes>
</Environment>
</Request>

**Figure 4.2 Example of an XACML Request Document**

In Figure 4.2, the first <Attributes> contains the attributes of the subject, who is making the access request. The subject has only one attribute that indicates his name as Omar.

The second <Attributes> contains the attributes of the resource to which the subject Omar has made the access request. The resource is identified by the resource's URI (/some/example.xml).
The third <Attributes> indicates that, sign is the action that the subject wants to perform on the requested resource. When the PDP receives the request context in Figure 4.2, it will locate the policy set in Figure 4.1 in its policy set repository. Hence, it compares the attribute values in the request context with the attribute values in the policy set's target. Since the attribute values in the policy set's target match the attribute values in the request context, the PDP will continue the evaluation.

Now, the PDP compares the attribute values in the request context with the attribute values in the target of the policy. Since the policy's target is empty, the policy matches this request context. Finally, the PDP compares the attribute values in the request context with the attribute values in the target of the only rule in the policy. Since the rule's target is omitted, the rule inherits the target form its parent, which is the policy element. Therefore, the rule's target matches the request context. Hence, the policy set evaluates to Permit.

A Response Context includes the authorization decision, which is obtained from the evaluation of the decision request against the policy by the PDP. It contains a sequence of one or more results in the <Result>. The <Response> contains one or more <Result> elements.

The <Result> includes the authorization decision from the PDP. The <Result> contains one <Decision>, zero or one <Status>, zero or one <Obligations>, zero or one <AssociatedAdvice>, and zero or more <Attributes> elements.

The required <Decision> includes the following authorization decisions:

1. Permit, if the requested access is permitted.
2. Deny, if the requested access is denied.
3. Indeterminate, if some error occurred during a policy set, policy, or rule evaluation process, such as missing attributes, network errors while retrieving policies or rules, division by zero during a policy or rule evaluation, or syntax errors in the decision request, in the policy, or in the rule.

4. NotApplicable, if no applicable policies or rules could be found.

If errors occurred during the evaluation of the decision request, the optional <Status> describes these occurred errors, and includes the information about them optionally.

The optional <Obligations> includes a list of obligations that must be fulfilled by the PEP. The <Obligations> contains one or more <Obligation> elements. The <Obligation> is composed of a set of attribute values that describe the action defined by the obligation. It contains an ObligationId attribute and zero or more <AttributeAssignment> elements. The required ObligationId attribute is an identifier for the obligation. The optional <AttributeAssignment> includes an attribute value, which is an argument retrieved from an obligation expression. It has a required AttributeId as an Identifier and an optional Category attribute (Scaglioso et al 2008, Rissanen 2013). An example of a response context is shown in Figure 4.3:

```xml
<Response>
  <Result>
    <Description> Permit </Description>
  </Result>
</Response>
```

Figure 4.3 Example of an XACML Response Document
4.3 DATA FLOW OF THE XACML MODEL

The data flow through the XACML model can be summarized as follows (Rissanen 2013):

1. The PAP writes policies and policy sets and makes them available to the PDP.

2. The requester sends an access request to the PEP module, which must execute the access decision that will be taken by the PDP.

3. The PEP module sends the access request to the context handler in its original format. It includes optionally, attributes of the subject, resource, action, and environment.

4. The context handler converts the original request into a canonical format, called the XACML request context, and sends it to the PDP.

5. The PDP requests any additional attributes of the subject, resource, action, environment, and other categories from the context handler.

6. The context handler makes queries to the PIP module to get the additional attributes.

7. The PIP interacts with the subject, resource, action, and environment modules, and provides the attribute values about them to the context handler.

8. The context handler sends the XACML request to the PDP.
9. 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.

10. The PDP evaluates the policies, and returns the XACML response context to the context handler.

11. The context handler converts the XACML response context to the original format of the PEP, and returns it to the PEP together with an optional set of obligations.

12. If the response context is Permit or Deny, the PEP fulfils the set of obligations.

13. If the access is permitted, the PEP performs the access. Otherwise, it denies access.

4.4 IMPLEMENTATIONS OF XACML

Sun implements an open source implementation of XACML in Sun's XACML (2004). The implementation was written in the Java programming language. This implementation supports parsing both policy and request/response documents, determining the applicability of policies, and evaluating requests against policies. All the standards attribute types, functions, and combining algorithms are supported, and there are APIs for adding new functionalities, as needed.

XACML.NET, which is available in "http://mvpos.sourceforge.net/download.htm", provides a graphical editor, called Control Center that allows the creating, editing, and evaluating of the XACML policies and requests. In this tool, the PDP component has been implemented in a pure .Net code (C#). It conforms to the XACML 1.0 specification, launched by OASIS (Mourad et al 2011).
The UMU-XACML-Editor was released by the University of Murcia (UMU) to assist in the creation and validation of the XACML policies. It is a graphic editor for policy definition, and provides a syntax-directed manner to construct a correct XACML policy. It has been implemented in Java programming language. It has been used a tree-based user interface design to ease the policy integration. Hence, it allows the users to edit and save policies in a friendly user interface, with the constraints of the XACML 2.0 specifications that were defined by OASIS (Mourad et al 2011).

XACMLight, which is available at http://sourceforge.net/projects/xacmllight/, is Axis2web service that implements a PDP and PAP, and defines all functions that are defined by the XACML 2.0 specifications.

4.5 LIMITATIONS OF XACML

The following are the limitations of XACML:

1. Although XACML is an integral policy description language, the structure of the existing XACML policy is very complex and hard. Therefore, it is necessary for the users to understand XACML well, in order to write all the securing policy specifications (Lang et al 2008).

2. The flexibility and expressiveness of XACML causes its complexity and verbosity. It is very difficult to work directly with XACML, especially if XACML is used to express high-level access control concepts, such as the separation of duties (Alm & Illig 2010).

3. XACML suffers from complexity and difficulty, and expertise is needed for writing all policy specifications by hand, which
limits its potential in supporting the security of web services. Moreover, a developer should have an expert in the business logic, security, policy specification languages, and web service technologies, to define the complex XACML constructs, such as policies, rules, targets, subjects, and actions, which is a cumbersome problem (Mourad et al 2011).

4.6 RBAC PROFILE OF XACML

Access control is concerned with authorizing a user to fulfil a specific task on a resource, such as a file, database object, or service (Decker 2008). This authorization decision is based on the organization's security purposes. The most commonly used model for modeling security is the RBAC model (Sandhu et al 1996). RBAC uses users' roles and the rules that determine the allowed accesses for users in their roles, to control the access to an organization. RBAC determines and implements security policies, which are mapped to an organization's structure. The RBAC approach satisfies security for organizations as long as they grant access only according to the organizational roles.

The basic concept of the RBAC model is that, users are assigned to roles, permissions are assigned to roles, and hence, users obtain permissions by being members of roles. The relationship between users and roles is a many-to-many relation; i.e., a user can be assigned to many roles and a role can have many users. Similarly, the relationship between permissions and roles is a many-to-many relation; i.e., a permission can be assigned to many roles and a role can have many permissions. The RBAC model has four levels, namely, the flat RBAC, hierarchical RBAC, constrained RBAC, and symmetric RBAC. The flat RBAC contains the basic requirement for RBAC. It has the requirement for user-role assignment revision. The hierarchical
RBAC has a requirement for supporting role hierarchies. The constrained RBAC adds a requirement for implementing the Separation of Duties (SoD). The Separation of duties distributes the responsibilities of a task over multiple users. There are two types of separation of duties, namely, static separation of duty (based on user-role assignment), and dynamic separation of duty (based on user-role activation). The symmetric RBAC adds a requirement for permission-role revision, similar to the flat RBAC. The advantage of the RBAC model lies in the easy and effective administration of security policies (Sandhu et al 1996).

The RBAC cannot specify any constraints on the location of the user, and the suitable time of his request to access a resource. Therefore, to limit the access of resources according to the location of the user and the suitable time of his request, OASIS has defined the RBAC profile of XACML, to add functionalities to limit the time and the location of the requester (Aburahma & Stumptner 2009). The RBAC profile of XACML satisfies the requirements for core and hierarchical role based access control. This profile does not need any extensions to the standard XACML (Anderson 2005).

4.6.1 Role

A role represents a job function or title within an organization. In the RBAC profile of XACML, roles are described as Subject attribute values in Role <PolicySet> and Permission <PolicySet>, or as Resource attribute values in a Role Assignment <PolicySet> or <Policy> and in a HasPrivilegesOfRole <Policy>. The role is described in two ways. The first way is depicted in Figure 4.4:
<Target>
  <Subjects>
    <Subject>
      <SubjectMatch MatchId="&function; string-equal">
        <AttributeValue
          DataType="&xml; string">
          manager
        </AttributeValue>
        <SubjectAttributeDesignator
          AttributeId="&role;"
          DataType="&xml; string"/>
      </SubjectMatch>
    </Subject>
  </Subjects>
</Target>

Figure 4.4 First Way of Describing Roles

The second way is depicted in Figure 4.5:

<Target>
  <Subjects>
    <Subject>
      <SubjectMatch MatchId="&function; string-equal">
        <AttributeValue
          DataType="&xml; string">
          manager
        </AttributeValue>
        <SubjectAttributeDesignator
          AttributeId="urn:someapp:attributes:manager-role"
          DataType="&xml; string"/>
      </SubjectMatch>
    </Subject>
  </Subjects>
</Target>

Figure 4.5 Second Way of Describing Roles
In the RBAC profile of XACML, there are four policies:

1. **Role <PolicySet> (RPS):** is a <PolicySet> that represents a role attribute and its value in the <Subject> of its <Target>. It companions the role with a Permission <PolicySet> that contains the permissions of the role. Each Role <PolicySet> can refer to a single Permission <PolicySet>, but it cannot contain or refer to any other <Policy> or <PolicySet> elements.

2. **Permission <PolicySet> (PPS):** is a <PolicySet>, which includes the permissions of a particular role. It contains the <Policy> and <Rule> elements, which describe the allowed resources for the subjects, and the actions that they can carry out on the resources. Moreover, they describe the conditions that must be satisfied to allow the subjects to access the resources. A Permission <PolicySet> may inherit all the permissions associated with other roles that are junior to the given role by refereeing to different Permission <PolicySet> elements associated with other roles. If the Permission <PolicySet> has a <Target>, it must not restrict the subjects to which the <PolicySet> is applicable. The Permission <PolicySet> depends on its Role <PolicySet> to be sure that, only subjects having the corresponding role attribute will have the permissions in the given Permission <PolicySet>.

3. **Role Assignment <Policy> or <PolicySet> (RAP):** is a <Policy> or <PolicySet> that assigns roles to subjects. A Role Enablement Authority (REA) uses this type of policy optionally.
4. HasPrivilegesOfRole <Policy>: is a <Policy> in a Permission <PolicySet> that checks whether a subject has the permissions associated with a given role or not.

A Permission <PolicySet> document must be stored in a way that it cannot be used as a policy for an XACML PDP. A Permission <PolicySet> is only ready through the corresponding Role <PolicySet> to be sure that, a Permission <PolicySet> must be allowed to every subject to provide hierarchical roles. Figures 4.6, 4.7, and 4.8 show examples of a Permission <PolicySet>, Role <PolicySet>, and Role Assignment <Policy> respectively (Anderson 2005):

```xml
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:policy:schema:os"
  PolicySetId = "PPS:manager:role" PolicyCombiningAlgId="&policy-combine; permit-overrides">
  <Policy PolicyId = "Permissions:for:the:manager:role"
    RuleCombiningAlgId = "&rule-combine; permit-overrides">
    <Rule RuleId = "Permission:to:update:salary file" Effect = "Permit">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch MatchId = "&function; string-equal">
              <AttributeValue
                DataType = "&xml; string">salary file</AttributeValue>
              <ResourceAttributeDesignator
                AttributeId = "&resource; resource-id"
                DataType = "&xml; string"/>
            </ResourceMatch>
          </Resource>
        </Resources>
        <Actions>
          <Action>
```
<ActionMatch MatchId="&function; string-equal">
  <AttributeValue
    DataType="&xml; string"> update </AttributeValue>
  <ActionAttributeDesignator
    AttributeId="&action; action-id"
    DataType="&xml; string"/>
</ActionMatch>
</Action>
</Actions>
</Target>
</Rule>
</Policy>
</PolicySet>

Figure 4.6 Permission <PolicySet> for a Manager

<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:policy:schema:os"
  PolicySetId="RPS:manager:role"
  PolicyCombiningAlgId="&policy-combine; permit-overrides">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="&function; string-equal">
          <AttributeValue
            DataType="&xml; string"> &roles; manager </AttributeValue>
          <SubjectAttributeDesignator AttributeId="&role;"
            DataType="&xml; string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <PolicySetIdReference>PPS:manager:role</PolicySetIdReference>
</PolicySet>

Figure 4.7 Role <PolicySet> for a Manager
The `<PolicySetIdReference>` associates the Permission `<PolicySet>`, named "PPS:manager:role" with the Role `<PolicySet>`, named "RPS:manager:role".

```xml
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:policy:schema:os"
        PolicyId="#Role:Assignment:Policy"
        RuleCombiningAlgId="#&rule-combine; permitoverrides">
    <Rule RuleId="#employee:role:requirements" Effect="#Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId="#&function; string-equal">
                        <AttributeValue
                            DataType="#&xml; string">Ahmed</AttributeValue>
                        <SubjectAttributeDesignator
                            AttributeId="#&subject;subject-id"
                            DataType="#&xml; string"/>
                    </SubjectMatch>
                </Subject>
                <Subject>
                    <SubjectMatch MatchId="#&function; string-equal">
                        <AttributeValue
                            DataType="#&xml; string">Mohammed</AttributeValue>
                        <SubjectAttributeDesignator
                            AttributeId="#&subject;subject-id"
                            DataType="#&xml; string"/>
                    </SubjectMatch>
                </Subject>
            </Subjects>
            <Resources>
                <Resource>
                    <ResourceMatch MatchId="#&function; string-equal">
                        <AttributeValue
                            DataType="#&xml; string">&roles; employee</AttributeValue>
                        <ResourceAttributeDesignator
                            AttributeId="#&role;"
                            DataType="#&xml; string"/>
                    </ResourceMatch>
                </Resource>
            </Resources>
        </Target>
    </Rule>
</Policy>
```

Figure 4.8 (Continued)
Figure 4.8 Role Assignment <Policy>
In Figure 4.8, the <Rule> indicates that, Ahmed and Mohammed are allowed to have the "&roles; employee" role enabled between the hours of 9 am and 5 pm. Moreover, the attributes of the role are described as Resource attribute values in the Role Assignment <Policy>.

In the RBAC profile of XACML, the REA is the component that assigns role attributes and values to users during a user's session. The REA assigns users to roles. It uses the XACML Role Assignment <Policy> or <PolicySet> to specify whether a subject is permitted to be a member of a particular role attribute, and value enabled or not. The REA may maintain a list of all roles in an organization and when asked about the role that can be assigned to a user, it makes a request to the Role Assignment Policies for each candidate role. If a user is allowed to be a member of multiple roles, any Role <PolicySet> instance applying to any of those roles may be evaluated, and the permissions from the corresponding Permission <PolicySet> will be permitted.

The XACML policies, using the RBAC profile of XACML, can use a <Condition> to set constraints on the application of a particular permission. By using a <Condition>, the RBAC profile can limit the permissions associated with a certain role, according to the location of the user, and the suitable time of his request.

**4.7 SUMMARY**

This chapter depicts the standard XACML and the RBAC profile of XACML. For the standard XACML, it depicts the rule, policy, and policy set of the XACML policy language. Moreover, it presents the XACML request and response language.

For the RBAC profile of XACML, it depicts the RBAC briefly and explains the role, and the types of policy of the RBAC profile of XACML with examples.