Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Semester Thesis

Access Control Markup Languages for XML Documents

Jan Rihak
jrihak@student.ethz.ch

Department of Computer Science
Swiss Federal Institute of Technology Zurich

August 2004

Supervisors:
Paul E. Sevinç
Dr. Bernhard Seybold
Prof. Dr. David Basin

Abstract

Access control is important in every enterprise. Resources are shared by different users, and access has to be controlled in a well-defined way. Therefore there is the need both for being able to formulate access control policies unambiguously and for mechanisms that enforce them.

The eXtensible Markup Language (XML) is an important standard for information exchange within and between enterprises. It is therefore not only interesting, but essential to study access control with respect to XML documents.

Several languages capable of formulating access control policies for XML documents have been proposed. This report presents and compares a few of these languages.

Contents

Chapter 1

Introduction

This chapter introduces the subject matter of access control markup languages for XML documents, motivates this report, and provides a short overview.

1.1 Motivation

Access control is important in every enterprise. In order for employees to properly do their job, they have to use resources which may be shared and ultimately belong to the enterprise. Ideally every employee has only the privileges absolutely necessary to fulfill his role, a principle called the least privilege principle.

According to [11], the eXtensible Markup Language (XML) [4],[12], a markup language maintained by the World Wide Web Consortium (W3C), has become the standard language for information exchange on the Internet. Web-based applications often rely on different access permissions on different parts of one and the same XML document. It is therefore important to provide access control not only on a file level, but also on portions of files, such as individual XML elements or subtrees.

Several languages exist that are capable of expressing access control at the granularity level of XML elements. This report examines the differences between some of these languages.

1.2 Overview

In chapter 2 we present four languages, namely XACL, Fine-Grained Access Control, XACML, and XrML. We state security requirements for an example use case and formulate access control policies in each of the four languages to formalize the security requirements. In chapter 3 we state criteria which we think are important to be fulfilled by access control policy languages for XML documents. We discuss how well the languages match these criteria. Chapter 4 concludes this report by summarizing chapters 2 and 3.

1.3 Related Work

A comparison of XACML and EPAL [17] can be found in [1]. Further comparisons seem not to have been made public.

Chapter 2

Access Control Markup Languages

In this chapter we present four markup languages for access control, namely the XML Access Control Language (XACL) [9], developed by the IBM Tokyo Research Laboratory, Fine-Grained Access Control [7], developed by the Università degli studi di Milano and the Polictecnico di Milano, the eXtensible Access Control Markup Language (XACML) [14], developed by OASIS, and the eXtensible rights Markup Language (XrML) [6], developed by the Xerox Palo Alto Research Center (PARC) and now maintained by ContentGuard. Before we do so, however, we provide an example use case whose security goals are confidentiality (no unauthorized read access) and integrity (no unauthorized write access).

2.1 Use Case

To illustrate the differences between the languages, we provide an example scenario with an object to protect and specific security requirements. In each of the four languages we write a policy that formalizes those requirements.

The objects to protect are medical records stored as XML documents. Every record contains administrative data and medical data. An example medical record is shown below:

```
<medicalRecord>
   <patient>
      <patientName>
         <first>Fredi</first>
         <last>Hinz</last>
      </patientName>
      <patientContact>
         <street>Gotthelfstrasse 14</street>
         <city>Zürich</city>
         <zip>8000</zip>
      </patientContact>
      <patient-number http://www.w3.org/2001/XMLSchema#type="integer">
         1234
      </patient-number>
   </patient>
   <primaryCarePhysician>
      <physicianID>
         5678
      </physicianID>
   </primaryCarePhysician>
   <medical>
      <illness>
```

```
         <name>leg fracture</name>
      </illness>
      <treatment>
         <drug>
            <name>antibiotics</name>
            <dailyDosage>2gs</dailyDosage>
            <startDate>2004-01-13</startDate>
         </drug>
      </treatment>
   </medical>
</medicalRecord>
```

A `<medicalRecord>` element contains three subelements, namely a `<patient>` element, a `<primaryCarePhysician>` element, and a `<medical>` element. The first two contain administrative data about the patient to whom the record refers and his primary care physician, respectively. The third element contains specific medical information about the illness of the patient and the treatment. We will refer to this example scenario record in later sections.

XML is used as a language to describe data and its structure. The elements contained in the document form hierarchies and have opening and closing tags. This structure enables the policy writer to address portions of an XML document like subtrees or sets of elements through path expressions, for example by using the path language XPath [22]. One can formulate different policies for different portions of a file. Of course, XML documents could as well be treated like any other document, where access decisions are made for the entire document.

2.2 Security Requirements

Subjects are identified by their name and organized in groups. We distinguish groups for patients, physicians, and the administrative staff. Of course, what we would like to express are different access rules for each of these groups. We require elementwise authorization for the elements in a medical record. The precise rules are stated below:

1. A patient can view an entire record, but only if it is his own.

2. A physician can view and modify any medical element of any record. Modifying an element means to alter the values of the element or subelements as well as appending additional subelements. Element modifications must be logged.

3. People working in the administration can view and modifiy administrative data such as addresses and phone numbers about patients and their physicians, but only during business hours from 9 am to 5 pm.

4. Any other access request is denied.

In the following sections we will formulate a policy expressing these security requirements in each language.

2.3 XML Access Control Language (XACL)

2.3.1 Introduction

XACL [9] is an XML-based language which was developed in 2000 by the IBM Tokyo Research Laboratory to provide fine-grained access control for XML documents. The policies defined in this language give the initiator of a request a specific view of the document and specific possibilities to securely update the document (or only parts of it), depending on the identifying attributes of the initiator. These attributes must properly authorize the initiator to perform the requested actions. With access-control policies written in XACL we are able to express element-wise or sub-tree-wise, temporal, data-dependent, or context-dependent authorization, etc.

2.3.2 Features of XACL

For details we refer to the language specification [9]. In this section we point out some important features of XACL.

XACL provides not only the possibility to specify which `subject` is authorized to perform what `action` on what `resource`, but it also gives the policy writer the possibility to specify `conditions` and `obligations`. A `condition` is a boolean formula returning "True" or "False" upon evaluation. `Conditions` pose further restrictions on an authorization decision, which might for example be temporal, location-dependent, data-dependent, or system-specific restrictions. An example for a temporal restriction is that access is only allowed during office hours. `Obligations` are actions which have to be performed upon a certain access decision. Logging a write action after the write was granted and executed is an example for an `obligation`. `Obligations` are called `provisional actions` within XACL, but we will use the term `obligation` throughout this report.

XACL supports policies for multiple resources, multiple subjects, and multiple actions. This makes it possible to formulate the same rule for a set of entities, which facilitates policy writing. Subjects can be further categorized into groups and roles [8].

Resources, which are specified through XPath [22] expressions can have a granularity as fine as individual XML elements. XPath gives us the possibility to access whole subtrees of an XML target document recursively, or to access a certain set of elements which all fulfill certain data-dependent conditions. Such a set of XML elements might be specified and matched through a restriction on some attribute values.

XACL is not extensible, and therefore it is not possible to define additional attributes by which a subject, resource, action etc. could be specified.

A policy is always associated with exactly one target XML document. It is not possible to define hierarchies of policies, and policy combination is not supported by the language.

An XACL installation consists of two main parts, an access evaluation module and a request execution module. The access evaluation module finds an appropriate policy for a given access request and returns an access decision as well as provisional actions. The request execution module updates the target document appropriately or generates the initiator's view of the document, given the access decision was "Grant". Any provisional actions attached to the policy will be executed. An XACL implementation is provided within the XML Security Suite [10], provided by IBM.

2.3.3 Example

Rule 1 The first rule translates into the following XACL code:

```
<?xml version="1.0"?>

<!DOCTYPE policy SYSTEM "policy.dtd">
<policy>
  <!-- =======================================================
  1. The "patient" group can read every element of a record,
     provided it is its own.
  ======================================================= -->
  <xacl>
    <object href="/medicalRecord/"/>
    <rule>
      <acl>
        <subject>
          <group>patient</group>
        </subject>
        <action name="read" permission="grant"/>
        <condition operation="and">
          <predicate name="compareStr">
            <parameter>eq</parameter>
            <parameter>
              <function name="getValue">
                <parameter>./patient/patient-number</parameter>
              </function>
            </parameter>
            <parameter>
              <function name="getUid"/>
            </parameter>
          </predicate>
        </condition>
      </acl>
    </rule>
  </xacl>
</policy>
```

Discussion of Rule 1 We define the XACL `<policy>` element according to a Document Type Definition [16]. The `policy` contains just one `<xacl>` element, even though the number of `<xacl>` elements is not bound. We will add further `<xacl>` elements to the same policy element when we discuss rules 2 to 4.

The first subelement of `<xacl>` is the `<object>` element which specifies a reference to every `<medicalRecord>`. The authorization will be applicable to the specified XML element and all its subelements. This is followed by the definition of a `<rule>` element. It is possible to define multiple `rules` all applicable to the same `objects`. The `rule` specifies the `<subject>` element. In our example this element defines that the issuer of the request must belong to the group `patient`. Furthermore, the action `read` is specified, and the `permission` attribute, which goes along with the `<action>` element, explicitly permits the action.

The `rule` can only evaluate to `"True"` if the `subject`, `resource` and `action` from the request context match with the policy and if the `condition` defined evaluates to `"True"` as well. In the `<condition>` element we use the `compareStr` function to compare the `<patient-number>` element in the target document with the `patient-number` of the `subject`. Since XACL defines a `subject` always as a triple of `<uid>`, `<role>` and `<group>` elements, we have to state the precondition that the `uid` corresponds to the `patient-number`, in case that

the `subject` is a member of the group `patient`. The `uid` of the `subject` making the request can be extracted with the function `getUid`.

Rule 2 Rule 2 translates into the following XACL code:

```
<!-- ====================================================
  2. A physician can view and modify any medical element
     of any record. Modifications have to be logged.
===================================================== -->
<xacl>
  <object href="/medicalRecord/medical/"/>
  <rule>
    <acl>
      <subject>
        <group>physician</group>
      </subject>
      <action name="read" permission="grant">
      <action name="write" permission="grant">
        <provisional_action name="log" timing="after"/>
      </action>
    </acl>
  </rule>
</xacl>
```

Discussion of Rule 2 Here we define a single `<xacl>` element. As already mentioned in the discussion of rule 1, we can simply add this element to the `<policy>` element defined in the XACL markup for rule 1. This `<xacl>` element specifies an `<object>` element, which simply refers to the `<medical>` element within a `<medicalRecord>` element of the target document.

Within the `<rule>` element we specify an `<acl>` element containing the information about the `subject` and the `action` which is to be granted on the specified `object`. The `<subject>` element states that the issuer of the request has to be a member of the group `physician`, and the two `<action>` elements specify the actions `read` and `write` to be granted. The action `write` has an additional `<provisional_action>` element attached to it, which states that when `write` access is granted, this access has to be logged after the execution of the `write`. The `<provisional_action>` element formulates `obligations` in XACL.

Rule 3 The third rule translates into the following XACL code:

```
<!-- ====================================================
3. People working in the administration can view and
   modifiy general data about patients and their
   physicians, like addresses and phone numbers, but only
   during business hours from 9 am to 5 pm.
===================================================== -->
<xacl>
  <object href="/medicalRecord/patient/"/>
  <object href="/medicalRecord/primaryCarePhysician/"/>
  <rule>
    <acl>
      <subject>
        <group>administration</group>
      </subject>
      <action name="read" permission="grant"/>
      <action name="write" permission="grant"/>
      <condition operation="and">
```

```
        <predicate name="timeInRange">
          <parameter><function name="getCurrentTime"/></parameter>
          <parameter>09:00 AM</parameter>
          <parameter>05:00 PM</parameter>
        </predicate>
      </condition>
    </acl>
  </rule>
</xacl>
```

Discussion of Rule 3 Just as we did for rule 2, for rule 3 we define a new `<xacl>` element which is added to the `<policy>` element defined in the markup of rule 1. The `<xacl>` element specifies two `objects`, namely the `<patient>` and the `<primaryCarePhysician>` element of the target document. The following `<rule>` element applies to both `objects`, therefore rule 3 is an example of XACL's capability of handling multiple resources.

In the `<rule>` element, we have again only one `<acl>` element. The `<acl>` element defines a `subject` which has to be member of the group `administration`. Further, two `action` elements are specified, `read` and `write`. In order for the `rule` to evaluate to `"True"`, the `condition` has to be satisfied. The `<condition>` element uses the `timeInRange` function to specify that the current system time has to be in between 9 am and 5 pm. The function `timeInRange` is not mentioned in the language specification [9] and would have to be defined and implemented separately.

Rule 4 The forth rule states that any other access request is denied. When none of the `<xacl>` elements in the policy elements apply and evaluate to `"True"`, then the default response of XACL denies the access.

2.4 Fine-Grained Access Control

2.4.1 Introduction

The access control system we present in this section was developed at the Università degli studi di Milano and the Polictecnico di Milano. Fine-Grained Access Control is a model for regulating access to XML documents and XML-based services, just as XACL is. The proposed approach exploits XML's own capabilities to define an XML markup describing the protection requirements of XML documents. This security markup can be used to provide both instance-level and schema-level authorizations with the granularity of XML elements/attributes .

2.4.2 Features of Fine-Grained Access Control

In this section, we discuss important language properties. For a more detailed introduction to Fine-Grained Access Control we refer to [7].

Authorization subjects Fine-Grained Access Control defines a `subject` as a triple of values, `<user-id,IP-address,sym-address>`, which expresses the user's identity, the IP adress he is issuing the request from and additionally the corresponding symbolic address. An example is `<john,192.128.0.0,john.company.com>`. The language also supports groups and

location patterns. A policy can therefore be applicable to multiple users with the same group membership and for users issuing the request from different physical locations within some domain. The language does not provide any means to extend the definition of `subject` or to add additional attributes.

Authorization objects As already mentioned, the resources to protect are not only files as a whole, but single portions of such files. Therefore, the `object` must be specified by a uniform resource identifier (URI) [2] and a path expression, in order to access single elements or attributes within the XML target document. For formulating path expressions, Fine-Grained Access Control uses the XPath language.

Features of access authorizations Fine-Grained Access Control supports authorizations at all levels of granularity, meaning XML documents as a whole as well as single elements and attributes within them. We can specify authorizations at the DTD level [16], [21], [3] (DTD- or schema-level authorizations), which implies that the policies we define are applicable to all instances of a certain DTD. But it is also possible to define authorizations for single XML documents (document- or instance-level authorizations), applicable to only one document. Fine-Grained Access Control provides a mechanism to combine different authorizations. This feature makes it possible to define a DTD-level authorization in order to state general security requirements, and an instance-level authorization in order to state specific security requirements applicable to just one target document.

Authorizations can be positive or negative to explicitly grant or deny access. Furthermore they can be local or recursive to address single XML elements or subtrees rooted at specific XML elements in the hierarchy. Possible conflicts of policies applicable to the same node are resolved by overwriting authorizations from a higher level in the tree by authorizations at a lower level, according to the “most specific takes precedence” principle. By default, this principle also applies to the combinations of DTD-level authorizations with instance-level authorizations. To override this principle, Fine-Grained Access Control provides possibilities discussed in the next paragraph. For when there are still conflicts and no proper precedence can be determined, there are means to formulate a conflict resolution policy.

Access authorizations and policy combination Access authorizations define what `subject` has access to what `object`. Access authorizations are formally defined as a five-tuple of the form `<subject, object, action, sign, type>`. The element `sign` indicates wether the authorization is a positive or a negative one. This makes it possible to explicitly forbid the access of a specific subject to specific objects. The `type` element is used to explicitly specify precedence relations of possibly conflicting authorizations. To override the “most specific takes precedence” principle we allow users to specify instance level authorizations as soft and DTD-level authorizations as hard. Soft authorizations at the instance level apply to the document unless some other authorization is stated at the DTD-level. Analogously, hard authorizations at the DTD-level apply to instances of this DTD in every case and cannot be overwritten.

Coverage Fine-Grained Access Control does not support obligations. Conditions can only be formulated data dependently through the use of XPath or location dependently through the specification of a physical location associated with the subject. Fine-Grained Access Control provides read actions and discusses how write actions could be supported. There exists a prototype implementation.

2.4.3 Example

As already mentioned, access authorizations in Fine-Grained Access Control have the form of a five-tuple <subject, object, action, sign, type>. Therefore, we will store the access authorizations in tabular form.

Rule 1 The first rule states that the "patient" group can read every element of a record, provided it is its own. This rule cannot be fully translated. The best we can do is shown in Table 2.1.

Subject user/group,IP,domain	Object (path expression)	Action	Sign	Type
Patient,*,*	/medicalRecord	read	+	R

Table 2.1: Fine-Grained Access Control Rule 1

Discussion of Rule 1 It is not possible to state the condition that the record has to belong to the patient. Although we could query the patient-number with an XPath expression, there is no possibility to access the patient-number attribute of the subject issuing the request. In the first column we state that the requesting subject has to belong to the group patient, but that we do not care about the IP address and domain of the subject. In the second column we express information about the object, which is a <medicalRecord> element of the target document. As action we specify read, and the sign indicates that the access to the object of the requesting subject shall be granted for the specified action. The type (R) finally tells us that the access decision not only applies to the single element <medicalRecord>, but also recursively to all subelements rooted at the <medicalRecord> element.

Rule 2 The second rule states that a physician can view and modify any <medical> element of any record. If an element has been modified, this change is logged. This rule cannot be fully translated. The best we can do is shown in Table 2.2

Subject user/group,IP,domain	Object (path expression)	Action	Sign	Type
Physician,*,*	/medicalRecord/medical	read	+	R
Physician,*,*	/medicalRecord/medical	write	+	R

Table 2.2: Fine-Grained Access Control Rule 2

Discussion of Rule 2 Fine-Grained Access Control does not support obligations. Our requirement to log changes to any <medical> element therefore cannot be expressed. What we can express is that the subject has to be a member of the group physician, and that this member has read and write access to the <medical> element within the <medicalRecord> element. With the type (R) we specify that we want to grant recursive access, which means that every subelement has the same authorization settings as the parent node.

Rule 3 The third rule states that people working in the administration can view and modifiy administrative data about patients and their physicians, but only during business hours from 9 am to 5 pm.

Since Fine-Grained Access Control does not support temporal conditions, this rule cannot be fully translated. The access authorizations are listed in Table 2.3.

Subject user/group,IP,domain	Object (path expression)	Action	Sign	Type
Administration,*,*	/medicalRecord/patient	read	+	R
Administration,*,*	/medicalRecord/patient	write	+	R
Administration,*,*	/medicalRecord/primaryCarePhysician	read	+	R
Administration,*,*	/medicalRecord/primaryCarePhysician	write	+	R

Table 2.3: Fine-Grained Access Control Rule 3

Discussion of Rule 3 We express four access authorizations, one for each `action/object` combination. We want the people of the administration to have `read` and `write` access to both elements `<patient>` and `<primaryCarePhysician>` within the `<medicalRecord>` element. Again, we define the `type` for this access to be recursive (R) in order to apply the same authorization for every subelement as well. As already mentioned, we are not able to express the temporal condition.

Rule 4 The fourth rule states that any other access is denied. We state this in Fine-Grained Access Control by giving a rule, which applies to any possible access request, but can be overwritten by other rules.

Subject user/group,IP,domain	Object (path expression)	Action	Sign	Type
,,*	/	read	-	RS
,,*	/	write	-	RS

Table 2.4: Fine-Grained Access Control Rule 4

Discussion of Rule 4 These two rules, one for each action, apply to any `subject` (specified through the (*) wildcard), and to any `object` in the objectspace by specifying the root of the resource tree along with the `type` (RS). This specifies that the rules apply to any child resources recursively (R), but have to be considered as soft (S), meaning that they will be overwritten by any rule which does not have the `type` soft. The negative `sign` specifies that access shall be denied.

2.5 eXtensible Access Control Markup Language (XACML) 2.0

2.5.1 Introduction

The purpose of XACML is the expression of authorization policies in XML against objects that are themselves identified in XML. XACML covers both an access control policy language and a request/response language. So besides defining who can do what and when by generating the corresponding policies, it is also possible to express access requests and responses in XACML.

The eXtensible Access Control Markup Language (XACML) version 1.0 [14] has been an OASIS standard since 2003. Improvements have been made to the language and incorporated in version 2.0 [15]. Although version 2.0 is still a draft, we will discuss this version in order to mention interesting new features like handling hierarchical and multiple resources [18], [19].

2.5.2 Features of XACML

For the exact details on XACML the reader should refer to the language specification [15]. In this report we discuss the main features of the language.

Policy nesting and evaluation Authorization requirements within XACML are expressed through `rules` and `policies`, where `<policy>` elements contain one or more `<rule>` elements. `<Policy>` elements can furthermore be organized into a hierarchical structure. The language provides mechanisms to combine individual `policy` results or `rule` evaluations in case that multiple different `policies` or `rules` are applicable to the same access request. This is done with combining algorithms already defined through the language or specified by the policy writer through extending the language.

The notion of inheritance is realized within XACML. `Policies` and `rules` can inherit from elements which are located higher up in the hierarchy. This facilitates the formulation of security policies to a great amount. For example, when the `target` element of a `rule` is left out, the `target` of the `policy` grouping the `rule` is chosen instead.

Conditions and obligations XACML is able to express `conditions` and `obligations`. To formulate sophisticated `conditions` and `obligations` a rich set of predefined functions is provided. Such predefined functions might be used to access the system time, to compare date values, to send an email, to write to a log file, etc. It is possible to formulate a variety of `conditions`, like temporal, system-dependent, location-dependent, or data-dependent `conditions`. The predicates within `conditions` can be logically combined to form arbitrarily complex expressions. `Obligations` can be arbitrarily complex as well.

Extensibility The language definition of XACML allows the extension of the language at several points. New attributes can be defined for subjects, resources, and actions. User-specific functions can be defined and implemented as well as policy- and rule-combining algorithms.

2.5.3 Example

In the following we formalize the security requirements stated in Section 2.2. For every rule, we write one XACML `policy`. We make use of some new features from XACML 2.0.

Rule 1 We recall our first rule: A patient can view an entire record, but only if it is his own. This translates into the following XACML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
  xmlns="urn:oasis:xacml:2.0:policy:schema:wd:12"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:xacml:2.0:policy:schema:wd:12
                      ../schemas/oasis-xacml-2.0-policy-schema-wd-12.xsd"
  xmlns:ctx="urn:oasis:names:tc:xacml:2.0:context"
  xmlns:md="http://www.medico.com/schemas/medicalRecord.xsd"
  PolicyId="XACML-SamplePolicy-1"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:2.0:
                      rule-combining-algorithm:deny-overrides">
  <Target/>
  <Rule RuleId="XACML-SampleRule1" Effect="Permit">
    <Description>
      A patient may read his or her designated record
    </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">
              patients
            </AttributeValue>
            <SubjectAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:
                           example:attribute:group"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
        <Resource>
          <ResourceMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              http://www.hospital.com/schemas/medicalRecord.xsd
            </AttributeValue>
            <ResourceAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:
                           resource:target-namespace"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ResourceMatch>
          <ResourceMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              /md:medicalRecord
            </AttributeValue>
            <ResourceAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ResourceMatch>
        </Resource>
      </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
              AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
    <Condition>
      <Apply 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
            AttributeId="urn:oasis:names:tc:xacml:2.0:example:
                         attribute:patient-number"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
                           function:string-one-and-only">
          <AttributeSelector
            RequestContextPath="//md:medicalRecord/md:patient/
                                md:patient-number/text()"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
      </Apply>
    </Condition>
  </Rule>
</Policy>
```

Discussion of Rule 1 At the top of the policy we declare a couple of namespaces. Afterwards we define the `PolicyId` attribute and the rule-combinging algorithm. We do not have to worry about the rule-combining algorithm since we have only one rule stated in the policy. We state an empty `<Target/>` element at the policy level, which has the effect that this `policy` will match every access request. In the following `<Rule>` element, we specify the `Subjects`, `Resources`, and `Actions` within the `<Target>` element of the rule. Within these elements we define which `subject` shall be able to perform what `actions` on what `resources`. The `rule` is only applicable to the request, when the values in these elements match those in the request context. In the `<Subjects>` element we require the requesting subject to be member of the group `patient`. This is done with a `SubjectMatch` element, which uses a built-in string-matching function and matches the string `"patient"` to the subject attribute of the requesting subject, identified through a certain string value. In the `<Resources>` element we make sure to address only those `resources` which are in the namespace of our hospital. This is done analogously as in the `Subjects` element with a matching function that compares a string value with an attribute from the request context. A second `<ResourceMatch>` element extracts the root element `<medicalRecord>` of every document. This is done with the predefined function `xpath-node-match`. A `read` permission is specified in the `<Actions>` element. If the policy grants access, the granted subject will have permission to read the whole document the request refers to.

The most interesting part, however, is contained within the `<Condition>` element. The function `string-equal` compares the attribute `patient-number` of the `subject` issuing the request with the `patient-number` defined in the requested XML document itself. Only when the `string-equal` function returns `"True"`, the `condition` is satisfied and access granted.

Rule 2 Let us recall the second rule: A physician can view and modify any medical element of any record. Modification of an element must be logged. This translates to:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
  xmlns="urn:oasis:xacml:2.0:policy:schema:wd:12"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:xacml:2.0:policy:schema:wd:12
                      ../schemas/oasis-xacml-2.0-policy-schema-wd-12.xsd"
  xmlns:ctx="urn:oasis:names:tc:xacml:2.0:context"
  xmlns:md="http:www.hospital.com/schemas/medicalRecord.xsd"
  PolicyId="XACML-SamplePolicy-2"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
                      rule-combining-algorithm:deny-overrides">
  <Description>
    Policy for any medical record in the
    http://www.hospital.com/schemas/medicalRecord.xsd namespace
  </Description>
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            http://www.hospital.com/schemas/medicalRecord.xsd
          </AttributeValue>
          <ResourceAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:
                         target-namespace"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ResourceMatch>
      </Resource>
    </Resources>
  </Target>
  <Rule RuleId="XACML-SampleRule-2" Effect="Permit">
    <Description>
       A physician can modify any medical element of any record in
       the http://www.hospital.com/schemas/medicalRecord.xsd namespace.
       Modification has to be logged.
    </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">
              physician
            </AttributeValue>
            <SubjectAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:
                           example:attribute:group"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
        <Resource>
          <ResourceMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              /md:medicalRecord/md:medical
```

```
            </AttributeValue>
            <ResourceAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ResourceMatch>
        </Resource>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">
              write
            </AttributeValue>
            <ActionAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
  </Rule>
  <Obligations>
    <Obligation
      ObligationId="urn:oasis:names:tc:xacml:example:obligation:log"
      FulfillOn="Permit">
      <AttributeAssignment
        AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:logfile"
        DataType="http://www.w3.org/2001/XMLSchema#string">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          www.hospital.com/log/medicalChanges.log
        </AttributeValue>
      </AttributeAssignment>
      <AttributeAssignment
        AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
        DataType="http://www.w3.org/2001/XMLSchema#string">
        <SubjectAttributeDesignator
          AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </AttributeAssignment>
      <AttributeAssignment
        AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
        DataType="http://www.w3.org/2001/XMLSchema#string">
        has modified the medical record of Mr. / Mrs.
      </AttributeAssignment>
      <AttributeAssignment>
        AttributeId="urn:oasis:names:tc:xacml:2.0:
                     example:attribute:patient-name"
        DataType="http://www.w3.org/2001/XMLSchema#string">
        <AttributeSelector
          RequestContextPath="//md:/medicalRecord/md:patient/
                              md:patientName/md:last"
          DataType="http://www.w3.org/2001/XMLSchema#string">
      </AttributeAssignment>
    </Obligation>
  </Obligations>
</Policy>
```

Discussion of Rule 2 Again, we first have a couple of namespace declarations, followed by the specification of the `PolicyId` attribute and the rule-combining algorithm. After a description of the policy, we specify a target for the policy. Since we want to address single XML elements of the target document, we make sure that our target document is from the proper namespace. This is done with a `<ResourceMatch>` element. Afterwards we specify the `<Rule>` element of the policy. The `<Target>` element of the rule contains a `<Subjects>`, `<Resources>`, and `<Actions>` element. In the `<Subjects>` element we restrict our rule to apply only to subjects which are members of the `physician` group. This is done with a `<SubjectMatch>` element. In the `<Resources>` element we now address a single XML element of our target document. We make use of the function `xpath-node-match`, which fetches us the `<medical>` element of the target document. The last element of the target, the `<Actions>` element, refers to the permission `write`. When everything applies and access is granted, a subject of the group `physician` will be permitted to write element `<medical>` of the requested medical record.

Finally, the `<Policy>` element has an `<Obligation>` attached. This element denotes certain actions which have to be executed depending on the evaluation of the policy. The attribute `FulfillOn` defines upon which evaluation result the obligation has to be fulfilled. In our case, if the authorization response is `Permit`, the action stated in the obligation will be executed. The `ObligationId` attribute states that the action to be executed shall be `log`. In a couple of `<AttributeAssignment>` elements we specify the parameters for the logging operation. We define the name and location of the logfile, the attributes indicating who modified whose record and some further text.

Rule 3 The third rule states that people working in the administration can view and modifiy general data about patients and their physicians, but only during business hours from 9 am to 5 pm. These requirements translate into the following XACML policy:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
  xmlns="urn:oasis:xacml:2.0:policy:schema:wd:12"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:xacml:2.0:policy:schema:wd:12
                      ../schemas/oasis-xacml-2.0-policy-schema-wd-12.xsd"
  xmlns:ctx="urn:oasis:names:tc:xacml:2.0:context"
  xmlns:md="http:www.hospital.com/schemas/medicalRecord.xsd"
  PolicyId="XACML-SamplePolicy-3"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
                      rule-combining-algorithm:deny-overrides">
  <Description>
    Policy for any medical record in the
    http://www.hospital.com/schemas/medicalRecord.xsd namespace
  </Description>
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            http://www.hospital.com/schemas/medicalRecord.xsd
          </AttributeValue>
          <ResourceAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:
                         resource:target-namespace"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ResourceMatch>
      </Resource>
```

```
    </Resources>
  </Target>
  <Rule RuleId="XACML-SampleRule-3" Effect="Permit">
    <Description>
      People working in the administration shall be able to view the general
      data about a patients and their physicians, like addresses and phone
      numbers, but only during business hours from 9 am to 5 pm.
    </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">
              administration
            </AttributeValue>
            <SubjectAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:
                           example:attribute:group"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
        <Resource>
          <ResourceMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              /md:medicalRecord/md:patient
            </AttributeValue>
            <ResourceAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ResourceMatch>
        </Resource>
        <Resource>
          <ResourceMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              /md:medicalRecord/md:primaryCarePhysician
            </AttributeValue>
            <ResourceAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ResourceMatch>
        </Resource>
      </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
              AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ActionMatch>
        </Action>
        <Action>
          <ActionMatch
```

```
            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              write
            </AttributeValue>
            <ActionAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
  </Rule>
  <Condition FunctionId="http://research.sun.com/projects/xacml/names/
                        function#time-in-range">
    <Apply
      FunctionId="urn:oasis:names:tc:xacml:1.0:
                  function:time-one-and-only">
      <EnvironmentAttributeDesignator
        DataType="http://www.w3.org/2001/XMLSchema#time"
        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
    </Apply>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">
      09:00:00
    </AttributeValue>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">
      17:00:00
    </AttributeValue>
  </Condition>
</Policy>
```

Discussion of Rule 3 In the `<Target>` element of the policy we specify the namespace of the target document. In the `<Target>` element of the rule we make further restrictions. First, in the `<Subjects>` element we require the subject to be a member of the group `administration`. In the `<Resources>` element we state two different resources. One referring to the `<patient>` element of the target document, the other referring to the `<primaryCarePhysician>` element. The rule gives equivalent permissions on both elements. In the `<Actions>` element we specify two actions, `read` and `write`.

In the `<Condition>` element we restrict the access to be granted only during business hours. In order to achieve this we have to use a non-standard function called `time-in-range`. This function requires three attributes, namely the current time and two time values defining the range in between which the current time has to lie.

Even when both the policy and the rule apply, access can only be granted if the condition evaluates to `True`.

Rule 4 We do not have to code Rule 4 explicitly. If a request does not apply to any of the Policies defined, the result `NotApplicable` is returned to the Policy Enforcement Point (PEP), which then denies the access request.

2.6 eXtensible rights Markup Language (XrML) 2.0

2.6.1 Introduction

The eXtensible rights Markup Language (XrML) [6] is an XML-based language for expressing rights and conditions for the use of digital resources.

XrML was designed to be comprehensive, generic, and precise. Comprehensive in the sense that is possible to express both simple and complex rights in a straightforward manner and at different stages of the workflow. Generic in the sense of being extendible and applicable to many different fields. And precise in the sense that expressions in XrML are not ambiguous.

XrML supports mechanisms for handling authenticity using digital signatures, specifying the level of trust as well as pattern matching through XPath. It is split into several parts, namely the core schema, the standard extension schema, the content extension schema and other domain-specific extensions. The core schema defines the semantics at the heart of XrML 2.0. The standard extension schema deals with concepts not included in the core schema but which are generally applicable to typical XrML usage scenarios. The content specific extension schema defines rights management concepts related to digital content of works such as books, music, and video.

2.6.2 Features of XrML

The most central elements in XrML are license, grant, principal, right, resource, and condition. The core schema defines principal, right, resource, and condition in an abstract way. These elements must be concretized in extensions of the core schema.

License The license is the top-level construct in XrML 2.0 and acts as a container of grants. It may contain additional information such as the identification of the principal or principals which issued the license, by whom the license additionally can be digitally signed.

Grant A grant is an element within the license element and is used to bestow an authorization upon a principal. This authorization defines a right against a resource and further conditions which must be met in order for the principal to be authorized.

Principal A principal corresponds to the subject to which rights can be granted. The principal can be identified with possibly multiple credentials which must be valid, as an owner of a public/private key pair, or some other identification technology.

Right A right defines an action or activity which can be executed by the principal. Rights can relate to other rights in the sense that a principal may be authorized to issue, revoke, delegate, or obtain other rights.

Resource A resource is the object to which a principal may be granted a right. A resource could be a digital work, a service or application, or even a piece of information owned by a principal (such as a name or an email address).

Condition A condition is used to specify further constraints on the execution of a defined right with respect to a resource. A condition can contain obligations, i.e., actions which have to be executed before or after the execution of the right.

2.6.3 Example

When trying to formulate our security requirements in XrML, we were faced with the following problems:

- We found that the language has difficulties expressing fairly simple properties like the group membership of a subject in a transparent way, or the restriction that an element of the target document has to match a certain attribute of the issuer of the request.
- The language forces the policy writer to fiddle with many details, like the authentication of the issuer of a license through digital signatures. Obligations are supported, but much of the complexity of their enforcement must be handled within XrML directly.
- XrML seems to be better suited to handle static resources, like ebooks, audio or video files, but has difficulties to handle dynamic resources, like modifiable XML documents.

XrML is suited for expressing licenses, not security policies. Although theoretically the same concepts can be formulated within licenses, when trying to do so, one is confronted with more difficulties than doing the same with security policies. Licenses are typically used to express simple rights a user might be granted, not complex conditional evaluations and obligations.

Chapter 3

Comparison

In this chapter we present the criteria that we think are important to be supported by access control languages for XML documents. The languages are evaluated and compared according to these criteria, and at the end of the section the results are summarized in a table.

We do not include XrML in our comparison. As we saw in chapter 2 XrML has a different scope than the other three languages.

3.1 Criteria

We compare the access control languages on the following criteria:

- Granularity
- Support for groups and roles
- Support for conditions
- Support for obligations
- Policy combination
- Predefined functions
- Exception and error handling
- Extensibility
- General flexibility
- Software support

3.2 Granularity

The granularity of the target resources can go as far as single elements or even attributes within Fine-Grained Access Control and XACML. Fine-Grained Access Control uses the path language XPath to specify the resource. XACML offers several possibilities to specify a resource, and in

particular with the predefined function `xpath-node-match` resources at the granularity of an XML attribute can be fetched.

XACL can apply access control policies to single XML elements, but not attributes.

3.3 Support for roles and groups

The support for roles [8] and groups facilitates policy writing to a great amount. It is also often necessary to be able to specify multiple subjects representing separate entities acting with different capabilities.

For XACL it is possible to specify group and role membership for every entity, as well as multiple subjects, although all subjects must be treated the same way. Fine-Grained Access Control is even coarser. It allows the policy writer to specify either a user or a group name, but roles are not supported. The only possibility to allow multiple subjects for the same policy is through groups or patterns of the numeric IP address or the symolic name of the address, so-called location patterns.

XACML supports roles and groups as well as multiple subjects. The language goes even further in providing the possibility to define arbitrary new attributes for subjects, not only those for group or role membership. The support for multiple subjects and their categorization with the `Subject-Category` attribute enables the system to base access decisions on the authorization of different users acting with different capabilities.

3.4 Support for conditions

The notion of conditions is supported in all languages, but to a different extent. The logical combination of predicates to form complex expressions out of simple ones is possible in all languages. The specification of XACL defines only few functions and predicates, which makes it difficult to formulate detailed temporal, location-, data- or system-dependent conditions. Fine-Grained Access Control supports only data-dependent conditions through the use of XPath and location-dependent conditions on the subject through location patterns. There are no possibilities to specify temporal or system-dependent conditions. XACML makes it possible to base condition evaluation on attributes of subjects, resources, or actions. A rich set of predefined functions enables the policy writer to evaluate system- or environment-specific values. Definition of own functions to define more specific conditions is possible.

3.5 Support for obligations

The notion of obligations is supported in XACL, where they are called provisional actions. However, the poor set of predefined functions and predicates poses constraints on the flexibility in defining those. Fine-Grained Access Control does not support obligations at all. XACML allows the policy writer to formulate obligations very flexibly, again with the use of a wealth of predefined functions.

3.6 Policy combination

With the possibility to organize policies hierarchically we need mechanisms which, in case some policies are applicable to the same access request, combine the individual access decisions to one single result. The notion of policy references makes it possible to manage policies centrally at one place, and reference and include them from other places. This facilitates decentralised policy management. Policy references are only supported by XACML.

In XACL for each XML target document exactly one policy can be defined. However, rules for single elements of this document can be specified and propagated upwards or downwards in the hierarchy, and combination algorithms can be specified.

With Fine-Grained Access Control it is possible to define several access control policies for the same document. This can be done by distinguishing between instance- and schema-level authorizations. The idea is that (several) policies can be attached to a schema, which are then applicable to all XML documents according to this schema. For every instance, only one instance-level policy can be attached. When nothing specific is said concerning conflicting policies, the “most specific takes precedence principle” is automatically applied. In case where these precendence criteria should not be applied, the policy writer has the possibility to define policies as hard or soft. Policies specified as hard will never be overwritten by other policies, where policies specified as soft will be overwritten by normal or hard policies in case of conflicts.

XACML provides extensive possibilities to organize policies and rules in hierarchies and the ability for policy referencing. Different conflicting policies and rules can be combined to give one single result by using a variety of predefined rule- and policy-combination algorithms. If needed, user-specific combination algorithms can be formulated to extend the language.

3.7 Predefined functions

An important issue is how much functionality is specified by the language itself, and how much of the functionality is left to be dealt with by the policy writer. It is easier to specify precise policies, when a policy writer is able to make use of a rich set of predefined functions and types. Examples might be the comparison of dates or the lexicographical comparison of strings. When a date or time value can only be specified as a string, an additional application would be necessary to parse the string into date or time format. It is easier to make use of a built-in comparison function, which can be addressed by a defined language construct and has a well-defined semantics, than implementing a new function and integrating it into the environment. Therefore we wish languages to provide a rich set of predefined functions and types.

Unfortunately XACL has only a poor set of such predefined functions. Fine-Grained Access Control can access only the functions offered through XPath to perform some operations and evaluations on the resources. Only XACML offers a large set of functions and types to be used directly by the policy writer. The functions supported in each of the three languages are listed in the specifications.

3.8 Exception and error handling

XACL applies a default policy in case where there is no authorization applicable for an access request. The specification does not discuss the possible occurrence of errors. Errors might be caused

through values which are not accessible or functions which cause evaluation errors. Exceptions of this kind would have to be handled in an undefined way by the implementing application.

Fine-Grained Access Control does not mention the handling of exceptions.

XACML specifies what the reaction in case of unapplicable policies or unaccessible attributes or values should be. Such policies evaluate to `NotApplicable` or `Indeterminate`, which can then be combined with possibly additional evaluation results to a `Permit` or `Deny` response.

3.9 Extensibility

Unfortunately, XACL and Fine-Grained Access Control are not extensible. In contrast, XACML is extensible at various locations. It provides extension possibilities to define new attributes, types, functions, obligations, combining algorithms, etc.

3.10 General flexibility

The languages offer different possibilities in specifying subjects, resources, actions, conditions, obligations, attributes, and functions. Since we already mentioned conditions, obligations, attributes, and functions, we discuss the flexibility in specifying subjects, resources, and actions in this section.

3.10.1 Subjects

In XACL, a subject contains an optional `uid` element, zero or more role names, and group names. It is not possible to provide a subject with additional application specific attributes, like for example age or gender. This would have to be specified in a condition.

In Fine-Grained Access Control a subject is specified as a three-tupel of `user-id`, `IP-address`, `sym-address`. No other attributes are allowed to make further restrictions about subjects. The `user-id`, however, is not restricted to single users, but could also be used to reference groups or roles.

In XACML arbitrary attributes for a subject are possible. These attributes can be extracted with the `<SubjectAttributeDesignator>` or `<AttributeSelector>` element. The possibility to specify multiple subjects and categorize them with a `<SubjectCategory>` element makes it possible to let them act in different capabilities for the access decision.

3.10.2 Resources

XACL does not offer much flexibility on the resources. An XACL policy has to be attached to a target document to which it applies, therefore there is a one to one mapping between policy and document. The only flexibility on resources concerns the subset of XML elements of the target document.

Within Fine-Grained Access Control XPath is used to specify resources. It is therefore not possible to formulate data-dependent restrictions on the target document in full generality. XPath provides a set of predefined functions to extract information from the target document and allows the formulation of conditional expressions which may be combined to build boolean expressions. Still, functionality is restricted to that provided by the XPath language.

XACML provides additional mechanisms to base an authorization decision on arbitrary characteristics of the resource. Attributes of the resource may be identified by the `<ResourceAttributeDesignator>` or the `<AttributeSelector>` element. Arbitrary attributes can be defined as an extension to the language.

3.10.3 Actions

In this paragraph we want to examine which actions are supported by the four languages and how language elements can be extended to define own application-specific actions.

The XACL specification supports four different actions, namely `read`, `write`, `create` and `delete`. The specification leaves no possibility to define individual actions.

Fine-Grained Access Control originally supports only `read` and `write` actions. `Write` actions should allow the insertion, update, and deletion of XML elements within the target document. The actions can be further restricted in Fine-Grained Access Control by data-dependent conditions specified in XPath. It is not possible to define individual actions as an extension.

In XACML it is possible to link arbitrary attributes with an `<action>` element, analogously to linking attributes to subjects and resources.

3.11 Software support

The XACL language is implemented as part of IBM's XML Security Suite [10]. The implementation does not follow the specification in every detail and leaves some functionality unimplemented. Fine-Grained Access Control does only provide a proof of concept through a demo application. For XACML various implementations are available, among them such from Sun Microsystems [13], Parthenon Computing [5], and Lagash Systems [20].

3.12 Comparison summary table

The differences between XACL, Fine-Grained Access Control, and XACML are summarized in Table 3.1.

Feature	**XACL**	**Fine-Grained AC**	**XACML**
Granularity	Elements, not attributes	Elements and attributes	Elements and attributes
Groups and roles	Supported	Supported	Supported
Support for conditions	Conditions supported, but low flexibility because of poor set of predefined functions and predicates	Only data-dependent and location-dependent conditions supported	Support for data-, location-, system-dependent, and temporal conditions through a large set of predefined functions and predicates
Support for obligations	Obligations supported through so called provisional actions, but with low flexibility because of poor set of predefined provisional actions	Obligations not supported	Obligations supported, high flexibility through large set of predefined functions
Policy combination	Exactly one policy possible for every target XML document. Rule hierarchies and rule combination supported. Policy referencing not supported	Hierarchies supported, various methods of combination supported. Policy referencing not supported	Hierarchies supported, various methods of combination supported, additional ones can be defined. Policy referencing supported
Predefined functions	Poor set	Poor set	Rich set
Exception and error handling	Not supported	Not supported	Supported
Extensibility	Not extensible	Not extensible	Extensible at various points
General flexibility	Restricted	Restricted	Possible to flexibly define attributes for subjects, resources actions, etc.
Software support	Available as part of IBM's XML Security Suite [10]	Only demo application available	Several implementations available

Table 3.1: Comparison Summary Table

Chapter 4

Conclusion and Summary

XACL and Fine-Grained Access Control brought access control to XML documents at the granularity of elements or even attributes (Fine-Grained Access Control). The languages are easy to understand, have compact and simple specifications, and enable the policy writer to formulate compact and simple policies. The main concepts of access control are supported by the languages, except that Fine-Grained Access Control does not support obligations at all and conditions not fully. Both of them are not extensible, so the policy writer has no means to extend the functionality of the languages. An implementation for Fine-Grained Access Control does only exist in the form of a demo application, and for XACL an implementation is only provided as part of IBM's XML Security Suite. These facts imply the limited applicability of the languages in the industry.

It has been seen that XrML is not suited for typical access control, but for Digital Rights Management (DRM). It provides authentication mechanisms, and suppports cryptographic elements such as digital signatures as a part of the language. The language was designed to be all-encompassing, is very complex and hard to grasp. Implementations are not freely available, the licensing terms are not clear, and the language focuses more on static resources (ebooks, audio, video) than on dynamic ones.

XACML is better suited to be applied in industry as an access control language for, but not only, XML documents. The language comes with a wealth of predefined functions, attributes, and types. However, the language specification and the policies are much more complex than those of XACL and Fine-Grained Access Control.

Bibliography

[1] A. Anderson. *A Comparison fo EPAL and XACML*. Sun Microsystems, Inc., 2004. Available at http://research.sun.com/projects/xacml/CompareEPALandXACML.html.

[2] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*. Available at http://www.isi.edu/in-notes/rfc2396.txt.

[3] P. V. Biron and A. Malhotra. *XML Schema Part 2: Datatypes*. World Wide Web Consortium (W3C), 2001. Available at http://www.w3.org/TR/xmlschema-2.

[4] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Third Edition)*. World Wide Web Consortium (W3C), 2004. Available at http://www.w3.org/TR/REC-xml.

[5] Parthenon Computing. *Parthenon XACML*. Available at http://www.parthenoncomputing.com/.

[6] ContentGuard. *eXtenible Rights Markup Language (XrML) 2.0*, 2001. Available at http://www.xrml.org.

[7] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. *A Fine-Grained Access Control System for XML Documents*. ACM Transactions on Information and System Security (TISSEC).

[8] D. Ferraiolo and R. Kuhn. *Role-Based Access Control*. 15th National Computer Security Conference, 1992. Available at: http://csrc.nist.gov/rbac.

[9] S. Hada and M. Kudo. *XML Access Control Language: Provisional Authorization for XML Documents*, 2000.

[10] IBM alphaWorks. *XML Security Suite*. Available at http://www.alphaworks.ibm.com/tech/xmlsecuritysuite.

[11] M. H. Kay. *XML five years on: a review of the achievements so far and the challenges ahead*. Proceedings of the 2003 ACM symposium on Document engineering, 2003.

[12] W. S. Means and E. Rusty Harold. *XML in a Nutshell*. O'Reilly, 2002.

[13] Sun Microsystems. *SUNXACML 1.2*. Availbale at http://sunxacml.sourceforge.net/.

[14] OASIS. *eXtensible Access Control Markup Language (XACML) Version 1.0*, 2003. Available at http://www.oasis-open.org/committees/xacml/repository/.

[15] OASIS. *eXtensible Access Control Markup Language (XACML) Version 2.0*, 2004. Available at http://www.oasis-open.org/committees/xacml/repository/.

[16] D. Raggett, A. Le Hors, and I. Jacobs. *Document Type Definition (DTD)*. World Wide Web Consortium (W3C), 1999. Available at http://www.w3.org/TR/REC-html40/sgml/dtd.html.

[17] M. Schunter, P. Ashley, S. Hada, G. Karjoth, and C. Powers. *Enterprise Privacy Authorization Language (EPAL 1.1)*. IBM Research, 2003. Available at http://www.zurich.ibm.com/security/enterprise-privacy/epal/.

[18] Sun Microsystems. *XACML Profile for Hierarchical Resources*, 2004.

[19] Sun Microsystems. *XACML Profile for Requests for Multiple Resources*, 2004.

[20] Lagash Systems. *XACML.NET*. Available at http://mvpos.sourceforge.net/.

[21] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema Part 1: Structures*. World Wide Web Consortium (W3C), 2001. Available at http://www.w3.org/TR/xmlschema-1.

[22] World Wide Web Consortium (W3C). *XML Path Language (XPath) 2.0*. Available at http://www.w3.org/TR/xpath20.