USING MICROSOFT OFFICE INFOPATH TO GENERATE XACML POLICIES

Manuel Sánchez, Gabriel López, Antonio F. Gómez-Skarmeta
Department of Information Engineering and Communications
University of Murcia, Spain

Óscar Cánovas
Department of Computer Engineering
University of Murcia, Spain

Keywords: Policy editor, XACML, XSLT, access control.

Abstract: Today, when organizations perform access control over their resources they are not only interested in the user's identity, but in other data such as user's attributes or contextual information. These requirements can be found, for example, in a network access control scenario where end users pay for a specific access level and depending on it, they can get different network quality of service. The network provider has to check, not only the user identity, but the user's attributes to make sure that he can access to the specified resource. These systems are based on the use of policy languages to define the authorization process. However, due to the increasing complexity of current systems, policies are becoming more and more complex to be managed by system administrators. Therefore, in this paper we present an user friendly approach to policy specification, based on the use of high level templates and common desktop applications. These templates are easily built from XML schemas, and once they have been filled, a XACML policy is automatically generated using a XML transformation.

1 INTRODUCTION

Access control management is a main concern for domain administrators since target resources have to be protected against unauthorized accesses from malicious attackers. One of the main scenarios is the network access control and several mechanisms have been proposed recently for that, such as 802.1X (IEEE Computer Society, 2001) or PANA (Forsberg et al., 2005). Initially, these mechanisms were only based on user authentication, mainly using shared secrets or public key cryptography. However, they are evolving to more sophisticated proposals performing user authorization based on user attributes, such as NAS-SAML (López et al., 2006). Usually, the authorization process is guided by a set of rules, that is, the access control policy, which has been usually described by means of specific, and technology dependent, policy languages, such as PERMIS (Chadwick et al., 2003) or Akenti (Thompson et al., 2003).

Due to the existence of different proprietary and application specific access control policy languages, policies can not be easily shared between different organizational domains, as described in (Cánovas et al., 2004), which involves a serious interoperability drawback. Besides, it makes difficult the development of good editing tools for access control policies. This situation has motivated the creation of a standard access control policy language, named XACML (Anderson et al., 2003), by the OASIS consortium. XACML, which was proposed to become a common framework for defining access control policies, addresses all the requirements of an access control environment and is currently used to represent and evaluate policies in a wide range of authorization scenarios, such as NAS-SAML (López et al., 2005) or systems described in (Lorch et al., 2002) (Cardea and PRIMA). However, this versatility implies a high level of complexity in the policy language definition. Therefore, administrators should have a depth knowledge of XACML in order to define an accurate medium-sized access control policy. For example, in a medium or large organization which needs to protect a high number of resources, policy management may become a very complex task.

Therefore, it would be desirable the existence of software tools to facilitate the development of access control policies from the point of view of the domain administrator, especially when they cannot be consid-

Sánchez M., López G., F. Gómez-Skarmeta A. and Cánovas Ó. (2006).
USING MICROSOFT OFFICE INFOPATH TO GENERATE XACML POLICIES.
In *Proceedings of the International Conference on Security and Cryptography*, pages 379-386
DOI: 10.5220/0002096803790386

ered XACML or security experts. At the moment there are several alternatives to deal with XACML policies, such as general XML editors or even specific XACML editors, but as explained below, all of them present any deficiency. This paper presents a framework to generate XACML policies from user friendly templates that can be filled in by a domain administrator in the same way they write a document in natural language, using the Microsoft®Office InfoPath™utility (Hoffman, 2003). Finally, the resulting high level policy files are transformed into the final low level XACML policies by means of XSL transformations (Clark, 1999).

In this way, as Figure 1 shows, to obtain a new XACML policy, the administrator writes the initial document. Then, the security expert of the organization applies the XML transformation to generate the policy. Finally, this policy is added to the XACML repository containing the rest of policies.

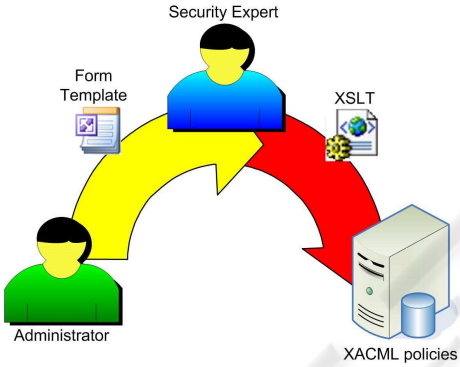

Figure 1: Policy life cycle.

The rest of this paper is structured as follows. Section 2 introduces the XACML standard. Section 3 explains the features of the tool used to design user friendly forms for administrators. The policy life cycle is shown in Section 4. To clarify the policy generation process, Section 5 presents a specific example. Related work is shown in Section 6. Finally, conclusions and future work are presented in Section 7.

2 XACML POLICY LANGUAGE

XACML (eXtensible Access Control Markup Language) (Anderson et al., 2003), the OASIS proposal for a standard access control language, was defined to represent access control policies in a standard way. XACML is XML-based and includes two different specifications: the first one is an access control policy language, which defines the set of subjects that can perform particular actions on a subset of resources; the second one is a representation format to encode access control requests and responses, that is, a way to express queries about whether a particular access should be allowed and the related answers.

As Figure 2 shows, the main element of all XACML policies is a *Policy* or *PolicySet* element. A PolicySet is a container that can hold other Policies or PolicySets, as well as references to other policies (*PolicyIDReference*). A Policy represents a single access control policy, expressed by a set of *Rules*. A Policy or PolicySet may contain multiple policies or Rules, each of which may evaluate to different access control decisions. XACML needs some way of reconciling the decisions each makes, and this is done through a collection of *Combining Algorithms*. An example of those is the *Permit Overrides Algorithm*, which says that if at least an evaluation returns Permit, then the final result is also Permit. A Policy or PolicySet element may also specify a set of *Obligation Attributes*, that is, a set of actions to be performed if the request has been approved.

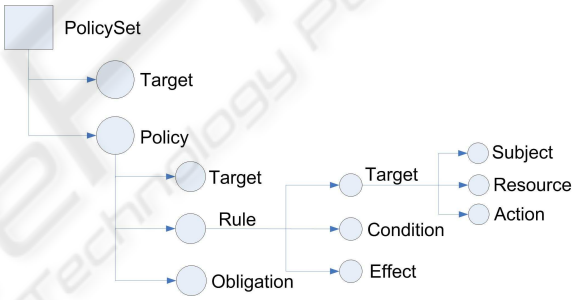

Figure 2: XACML policy structure.

XACML provides another feature called *Target*. It is a set of simplified conditions for the Subject, Resource and Action that must be met for a PolicySet, Policy or Rule to apply to a given request. If all the conditions of a Target are met, then its associated PolicySet, Policy, or Rule are applied to the request. Once a Policy is found, the included rules are evaluated. The main element of a rule is the *Condition*. If the evaluation of the Condition results true, then the Rule's Effect (Permit or Deny) is returned.

The main object that XACML deals with is attributes. Attributes are named values of known types. Specifically, attributes are characteristics of the Subject, Resource, Action or Environment in which the access request is made.

3 MICROSOFT OFFICE INFOPATH

InfoPath (Hoffman, 2003) is a desktop application which allows organizations to efficiently gather the needed information using dynamic forms. This architecture includes design characteristics which make easy the structured creation and flexible viewing of XML documents. This tool has the following architectural features and design objectives:

- Build an hybrid tool which combines document editing experience with the data capture ability of forms. Users can view and modify abstract data structures using a traditional word-processing environment.
- The use of XML documents belonging to user defined schemas for the input and output. InfoPath uses and produces XML schemas and XSL transformations, and it is integrated with XML Web services standards. When editing an XML document, InfoPath enables adding and deleting valid attributes and XML elements which belong to a XML schema defined by the user. Then, when the XML document is saved or submitted, it remains valid following the XML schema.
- Provide structural editing. InfoPath enables gathering structured and validated XML information which can be reused. Structural editing in InfoPath provides an easy and natural user's interface which allows to normal users add and delete valid XML attributes and elements.
- Provide flexible views to present XML documents in a coherent way to the user. InfoPath uses XSLT to enable the content of the editing views to be organized in a different way that the XML data structure.

Three different validation levels exist in InfoPath: schema based validation, XPath validation rules and script based validation. The first validation level helps users to create structured XML documents which are ready to be reused by XML based systems. The second level allows to check conditions, e.g. the value of some field must be greater than a particular minimum. And the third level may define additional validations in the XML document or define other business logic.

Form templates can be designed in several ways, ranging from an existing XML schema or document to defining an XML schema from scratch.

4 USING TEMPLATES TO GENERATE POLICIES

Complexity in policy definition languages makes difficult the specification of the policies governing the access control system. Therefore, a methodology to simplify this task is proposed in this document. The main idea, shown in Figure 3, is that the administrator writes a document in natural language expressing what the system must enforce, that is, the set of access control rules. Later, by means of XSL transformations, this document is translated into an XACML policy. Specifically, a set of templates are available to the system administrator, each one for every specific policy existing in the system. These templates define documents that can be used to describe a specific feature of the access control system, such as the roles assigned to each user or the set of roles enabled to access to a specific resource.

Figure 3: Policy generation.

Microsoft Word might be used as text editor to design the required templates, but InfoPath has been selected since this tool was specifically designed to work with templates. The design of a policy template is based on a XML schema defining the information elements which can be specified by the administrator. This schema should be designed by an XACML expert, since it has to contain the more relevant elements of the XACML policy.

Once the initial document is written, a XSL transformation must be applied to transform it into the final XACML policy. This can be done either using a XSLT compiler, such as XALAN (Apache Software Foundation, 2006), or some XML editor implementing the XSLT standard, such as XMLSpy (Altova, 2006). The XSL transform generates the XACML policy completing the required sections using the information from the initial document. As the initial XML schema, the design of the XSLT implies a deep knowledge about the XACML standard, because it is necessary to know how to generate a complete policy fulfilling the XACML specifications.

Consequently, the *"security expert"* is responsible for developing the XML schemas, the policy templates and the XSL transformations for each policy in the organization. On the other hand, specific system administrator, for example administrators who are in

charge of the role assignment to users, are able to specify the security policy without any knowledge of XACML. All they have to do is to complete the policy template, expressed in natural language, and this will be automatically transformed into the target XACML policy by means of the XSL transformations. Finally, policies are stored in the XACML repository to make them available to the system.

This kind of policies doesn't have real time requirements, therefore it is not necessary for administrators to apply the XSLT transformation and to store the XACML policy in the repository directly. In this way, administrators only have to fill in the template due to the security expert can perform all the related technical tasks.

The conversion of the XML document into the XACML policy depends mostly on the specific policy being generated, therefore transformation details are given next using some examples.

4.1 Translating Policy Elements

Definition of the XML schema containing the elements managed by the administrator can be made using the string data type, for attributes, subjects, etc. However, there are special cases needing special attention. For example, obligations commonly express the assignment of a value to an attribute, so they need to be represented as a pair of strings.

A more complicated example is related to *conditions*, because they can be expressed in a variety of ways. For example, $((: current - time \geq 9h) and (: current - time \leq 21h)) and ((: dayOfWeek \geq Monday) and (: dayOfWeek \leq Friday))$ needs to be represented in a recursive way, but InfoPath does not allow recursive schemas.

Therefore an iterative approach must be taken, where simple relational checks are joined using boolean operators, and new checks are joined to the previous relation as a whole. For example, the condition stated before could be expressed as $(((: current - time \geq 9h) and (: current - time \leq 21h)) and (: dayOfWeek \geq Monday)) and (: dayOfWeek \leq Friday)$. Moreover, since the most common condition in policies implies two relational comparison in the form of $(x > A) and (x < B)$, this structure can be introduced in the schema to make easy the design of the *Condition* element. Figure 4 shows a simplified view of the *Condition* element schema.

Finally, the basic XML document must be transformed into the complete XACML policy by means of a XSL transformation. Therefore, elements from the basic XML document must be translated into XACML elements, obeying the specific structure, formats and namespaces. The XSL transformation generates the specific policy structure using the *xsl:element* and *xsl:attribute* operators in a fixed way, and then it uses the input XML document generated before to fill in the specific gaps in the final XACML policy. Once the transformation is designed, it has only to be applied to every InfoPath-generated document.

5 POLICY GENERATION FOR A NETWORK ACCESS CONTROL AUTHORIZATION SCENARIO

This section depicts how to generate one of the required policies for the NAS-SAML system, where the main resource to be protected is the communication network.

5.1 Nas-saml. A Network Access Control Approach Based on Saml and XACML

NAS-SAML (López et al., 2006) is a network access control approach based on X.509 identity certificates and authorization attributes. This proposal is based on the SAML and the XACML standards, which will be used for expressing access control policies based on attributes, authorization statements and authorization protocols. Authorization is mainly based on the definition of access control policies including the sets of users pertaining to different subject domains which will be able to be assigned to different roles in order to gain access to the network of a service provider, under specific circumstances. The starting point is a network scenario based on the 802.1X standard and the AAA (Authentication, Authorization and Accounting) architecture, where processes related to authentication, authorization, and accounting are centralized.

The system operates as follows. Every end user belongs to a home domain, where he was given a set of attributes stating the roles he plays. When the end user requests a network connection in a particular domain by means of a 802.1X connection, the request is obtained by the AAA server, and it makes a query to obtain the attributes linked to the user from an authority responsible for managing them, based on a *Role Assignment Policy*. In case the user's home domain is based on a different authorization system, the AAA server uses a credential conversion service, as defined in (Cánovas et al., 2004), to translate the user's authorization credentials into internal format, based on a *Conversion Policy*. Finally, the AAA server sends an authorization query to a policy decision point (PDP), which consults a *Resource Access Policy*. The PDP firsty recognizes if the user belongs to well known

```
<xs:element name="Condition">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="SelectCondition"/>
   <xs:element ref="BooleanCondition"
     minOccurs="0" minOccurs="unbounded">
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="BooleanCondition">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="BooleanOperator"/>
   <xs:element ref="SelectCondition"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="SimpleCondition">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="LeftElement" type="xs:string"/>
   <xs:element name="RelationalOperator"/>
   <xs:element name="RightElement" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="DoubleCondition">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="SimpleCondition"/>
   <xs:element name="BooleanOperator"/>
   <xs:element ref="SimpleCondition"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="SelectCondition">
 <xs:complexType>
  <xs:choice>
   <xs:element ref="SimpleCondition"/>
   <xs:element ref="DoubleCondition"/>
  </xs:choice>
 </xs:complexType>
</xs:element>
```

Figure 4: Simplified Condition Element XML Schema.

source domain, by means of the *Role Allocation Policy* sub-policy. Then, it checks if the user is allowed or not to use the target resource, by means of the *Target Access Policy* sub-policy. Furthermore, it also establishes the set of obligations derived from the given decision, for example some QoS properties, security options, etc. This general scheme works both in single and inter-domain scenarios, and using both push an pull based communications.

5.2 Editing the Target Access Policy

This section shows an example of how one of the policies introduced in NAS-SAML (*Target Access Policy*), can be defined by means of the proposed policy editor. First, the *Target Access Policy* is described, and next the policy template is generated step by step.

5.2.1 Target XACML Policy

As described in (López et al., 2005), the *Target Access Policy* comprises a set of *target access* elements. Each of them grants a user playing a specified set of roles the permission to carry out the specified actions on the specified list of targets, but only if the are satisfied conditions. Every *target access* element has the following elements:

- *Attributes:* Set of allowed attributes or enabled roles to execute the actions on the resource.
- *Resources:* Set of controlled resources.
- *Actions:* Set of allowed actions over the resource.
- *Conditions:* Users holding some of the attributes have permission to execute some actions on the specified resources only if the conditions are fulfilled. Otherwise, the permission will be denied. These conditions can establish time constraints or other constraints related to contextual information.
- *Obligations:* Once the action has been granted, some obligations defining network properties might be applied. Obligations can specify options related to network addressing, security or QoS properties that must be enforced.

Figure 6 shows an example of the *Target Access Policy*. In this figure we can see a *PolicySet* element, where *Target* defines the role type and value, in this case *role-id* and *Student*. Besides, it contains a *Policy* to define that the *wireless-network* resource, can be *used* by the users playing that role. This example also shows the set of obligations derived from this decision. That is, if a user holding the *Student* role, requests *wireless-network* connection, it should obtain an IPv6 address from the range *2001:720:1710:100::/64* and the network must guarantee a quality of service established by *Class1*.

5.2.2 Definition of the XML Schema

To define the initial XML schema, we have to identify first the policy elements the administrator needs to complete in order to create a new policy instance. In this example, as described above, these elements are the attributes, the rights which are going to be granted, the resource/access pair, the conditions which have to be fulfilled to apply the policy decision and the obligations which may be derived. In this way, we can define an XML schema where an *Attribute* and several *Resource*/*Action* pairs are specified. Besides, *Condi-*

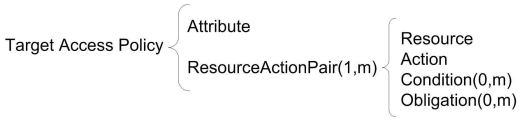

Figure 5: XML schema representation.

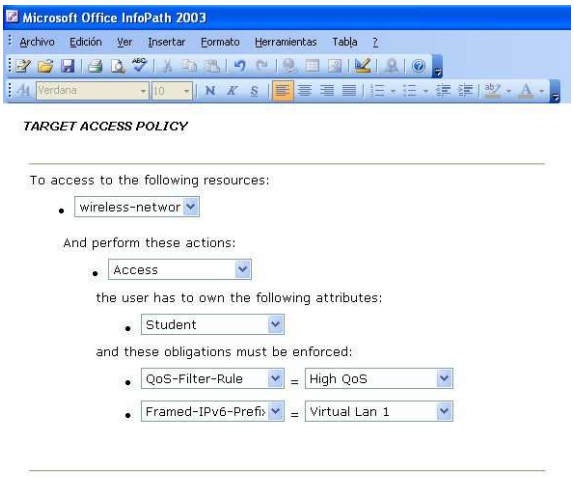

Figure 7: Target access policy form template.

tion and *Obligation* need to be stated as optional elements. Figure 5 clarifies this structure.

5.2.3 Definition of the XSL Transformation

The next step in the policy template generation process is the definition of the XSL transformation, which has to translate the document in natural language written by the system administrator into the final XACML policy. The resulting XACML policy is structured using a *PolicySet* to specify the *Attribute*. Then, for each resource/action pair, a new *Policy* is added with the *Resource*, *Action*, *Condition* and *Obligation* elements. The XSL transformation creates the XACML policy in a fixed way and then fills in some specific gaps using the information contained in the initial document.

Figure 8 shows a fragment of the XSL transformation used to generate the target access policy in this example. It is interesting to see how the XACML police structure is generated in a fixed way using the *xsl:element* and *xsl:attribute* operators, while the suitable gaps are completed using the sentence *xsl:value-of select="text()"* to extract the information from the initial XML document.

5.2.4 Design of the InfoPath Form Template

The final step in this process lies in the design of the InfoPath template to enable the system administrator an easy way to create new policies without XACML knowledge. This is done with the InfoPath editor, which allows to import the XML schema defining the template. Figure 7 shows an example of template designed using this editor.

6 RELATED WORK

This section analyzes two main tools which are currently used to manage XACML policies, XMLSpy(Altova, 2006) and UMU-XACML-Editor (University of Murcia, 2006). The first one is a powerful and commercial generic XML editor, the second one is a specific XACML editor developed by the University of Murcia.

XMLSpy is an IDE designed to work with XML documents in general, which allows users to enter data into XML documents as they would into a word processor-type application. It allows the definition of XML documents in multiple editing formats, well-formedness checking and built-in validator, intelligent editing, definition of XSLT documents, code generation, etc. These characteristics enable users to specify the XACML schema in the IDE and thus generate XACML policies from scratch in a fast way. The user builds the policy adding each element step by step and checking that it is well formed.

UMU-XACML-Editor is a XACML policy definition open source software, developed using Java, which fulfills the XACML 2.0 standard. This editor is specifically designed to create and modify XACML policies, providing facilities to manage these policies. It is free and can be downloaded from the OASIS's XACML Home Page (OASIS, 2006).

Although both solutions deals with XACML in a proper way, the main problem is that they work directly with XACML policies. In this way, the system administrator has to known the XACML standard to create a new policy. But also, he has to decide how to organize the elements in the XACML structure to express correctly the meaning of the policy. With our proposal, this work is made only once for each kind of policy. Besides, only one person, the security expert, has to deal with the XACML related work.

Furthermore, each XACML editor has its own editing interface and its own characteristics, so the system administrator has to learn how to use these programs before start editing policies. However, we provide standard templates to create the XACML policies. Therefore, the system administrator opens the template and completes it as he was writing a natural

```
<PolicySet PolicySetId="TargetAccessPolicy">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch>
          <AttributeValue>Student</AttributeValue>
          <SubjectAttributeDesignator AttributeId="role-id"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <Policy PolicyId="WirelessPermission">
    <Target/>
    <Rule RuleId="ruleid" Effect="Permit">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch>
              <AttributeValue>wireless-network</AttributeValue>
              <ResourceAttributeDesignator AttributeId=":resource-id"/>
            </ResourceMatch>
          </Resource>
        </Resources>
        <Actions>
          <Action>
            <ActionMatch>
              <AttributeValue>enable</AttributeValue>
              <ActionAttributeDesignator AttributeId="action-id"/>
            </ActionMatch>
          </Action>
        </Actions>
      </Target>
    </Rule>
    <Obligations>
      <Obligation ObligationId="obligation-id" FulfillOn="Permit">
        <AttributeAssignment AttributeId="Framed-IPv6-Prefix">
          2001:720:1710:100::/64</AttributeAssignment>
      </Obligation>
      <Obligation ObligationId="obligation-id" FulfillOn="Permit">
        <AttributeAssignment AttributeId="QoS-Filter-Rule">
          Class1</AttributeAssignment>
      </Obligation>
    </Obligations>
  </Policy>
</PolicySet>
```

Figure 6: Target Access Policy example.

```
<xsl:stylesheet>
  <xsl:output method="xml"/>
  <xsl:template match="WordTargetAccessPolicy">
    <xsl:element name="PolicySet">
      <xsl:attribute name="PolicySetId">
        GeneratedAccessPolicySet
      </xsl:attribute>
      <xsl:attribute name="RuleCombiningAlgId">
        permit-overrides
      </xsl:attribute>
      <xsl:element name="Description">
        Descritpion of the policy
      </xsl:element>
      <xsl:element name="Target">
        <xsl:element name="Subjects">
          <xsl:apply-templates select="Attribute"/>
        </xsl:element>
        <xsl:element name="Resources">
          <xsl:element name="AnyResource"/>
        </xsl:element>
        <xsl:element name="Actions">
          <xsl:element name="AnyAction"/>
        </xsl:element>
      </xsl:element>
      <xsl:apply-templates select="ResourcePolicy"/>
    </xsl:element>
  </xsl:template>

  <xsl:template match="WordTargetAccessPolicy/Attribute">
    <xsl:element name="Subject">
      <xsl:element name="SubjectMatch">
        <xsl:attribute name="MatchId">
          urn:oasis:names:tc:xacml:1.0:function:string-equal
        </xsl:attribute>
        <xsl:element name="AttributeValue">
          <xsl:attribute name="DataType">
            http://www.w3.org/2001/XMLSchema#string
          </xsl:attribute>
          <xsl:value-of select="text()"/>
        </xsl:element>
        <xsl:element name="SubjectAttributeDesignator">
          <xsl:attribute name="AttributeId">
            role-id
          </xsl:attribute>
          <xsl:attribute name="DataType">
            http://www.w3.org/2001/XMLSchema#string
          </xsl:attribute>
        </xsl:element>
      </xsl:element>
    </xsl:element>
  </xsl:template>
  . . .
  </xsl:template>
. . .
</xsl:stylesheet>
```

Figure 8: XSL transformation fragment.

language specification. He does not need any previous knowledge to start editing the policies of the organization. He only has to know which policy wants to define and then writes the appropriate document. Later, the security expert will deal with the automatic translation from the natural language document to the XACML policy.

Finally, related to modifications in policy format, when using a generic XML editor we have to specify the new XACML schema to generate new policies. On the other hand, with the specific UMU-XACML editor, the source code has to be modified to add the new elements and relations.

7 CONCLUSIONS AND FUTURE WORK

XACML has appeared to cover the need of the current authorization systems to represent in a standard language the access control policies used to control critical resources. Most of those current systems are migrating their specific policies to XACML in order to offer a more scalable and extensible solution.

The fact that XACML is a very helpful tool for secure domain administrator does not imply that it is easy to use. In a typical access control scenario, during a routine policy creation or update, the domain administrator does not have to deal with complex XML schemes and documents but only with the fulfillment of the set of permissions assigned to users under specific circumstances. In fact, the domain administrator could not even understand the XACML language.

It shows the need of tools able to help domain administrators to deal with complex XML documents in a natural language. Moreover, this paper shows that current generic XML or specific XACML editors do not fulfill these requirements.

We have defined a way to manage those XML documents in a transparent way for the domain administrator, making use of a word-processing style editor such as Microsoft InfoPath. In this way, once the necessary XML templates and transformations are created by a XACML expert, the domain administrator can define low level XACML policies using human readable forms.

The sample scenario we have used to test the proposed solution is the NAS-SAML infrastructure. This paper shows how the *Target Access Policy* template and its associated XSL transformation can be generated by a security expert to make easy the administration tasks to the network administrator. The rest of policies used in this scenario can be defined in a similar way.

Finally, the solution proposed in this work can be also used to define other kind of documents based on other XML specification, such as (Thompson et al., 2003; Chadwick et al., 2003).

REFERENCES

Altova (2006). XMLSpy®. `http://www.altova.com/xmlspy`.

Anderson, A., Parducci, B., Adams, C., Flinn, D., Brose, G., Lockhart, H., Beznosov, K., Kudo, M., Humenn, P., Godik, S., Andersen, S., Crocker, S., and Moses, T. (2003). *EXtensible Access Control Markup Language (XACML) Version 1.0*. OASIS Standard.

Apache Software Foundation (2006). The apache xalan project. `http://xalan.apache.org`.

Chadwick, D., Otenko, O., and Ball, E. (2003). Implementing role based access controls using x.509 attribute certificates. *IEEE Internet Computing*, pages 62 – 69.

Clark, J. (1999). *XSL Transformation (XSLT)*. W3C Recommendation.

Cánovas, O., Lopez, G., and Gómez-Skarmeta, A. (2004). A credential conversion service for saml-based scenarios. In *Proceedings First European PKI Workshop*, volume 3093 of *Lecture Notes in Computer Science*, pages 297–305. Springer.

Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and Yegin, A. (2005). *Protocol for Carrying Authentication for Network Access (PANA)*. Internet Draft.

Hoffman, M. (2003). *Architecture of Microsoft Office InfoPath 2003*. Microsoft Technical Report.

IEEE Computer Society (2001). P802.1x/d11: Standard for port based network access control. IEEE Draft.

Lorch, M., Proctor, S., Lepro, R., Kafura, D., and Shah, S. (2002). *First Experiences Using XACML for Access Control in Distributed Systems*. ACM Workshop on XML Security.

López, G., Cánovas, O., and Gómez, A. F. (2005). Use of xacml policies for a network access control service. In *Proceedings 4th International Workshop for Applied PKI, IWAP 05*, pages 111–122. IOS Press.

López, G., Cánovas, O., Gómez, A. F., Jimenez, J. D., and Marín, R. (2006). A network access control approach based on the aaa architecture and authorzation attributes. *Journal of Network and Computer Applications JNCA*. To be published.

OASIS (2006). OASIS eXtensible Access Control Markup Language (XACML) TC. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`.

Thompson, M., Essiari, A., and Mudumbai, S. (2003). Certificate-based authorization policy in a PKI environment. *ACM Transactions on Information and System Security (TISSEC)*, 6:566 – 588.

University of Murcia (2006). UMU XACML editor. `http://xacml.dif.um.es`.