XML-based Distributed Access Control System

Javier López, Antonio Maña, Mariemma I. Yagüe

Computer Science Department
University of Málaga. Spain.
{jlm, amg, yague}@lcc.uma.es

Abstract. Extensions of the role based access control (RBAC) paradigm have been proposed based in the use of attribute certificates. In order to overcome some of the limitations of RBAC and be able to implement security requirements such as the "originator controlled" (ORCON) policy, the concept of mobile policy has been recently prop osed. Mobile policies are attached to the data that they control and are enforced by their execution in trusted servers. In this paper we present an extension of this idea that allows the execution of the policies in untrusted systems. We also extend the scheme to allow the policies to be linked to the data but not attached to it. By this modification security administrators are able to change policies dynamically and transparently. Finally, we introduce *X-ACS*, a XML-based language designed to express policies in a simple and unambiguous way overcoming the limitations of other approaches. Important features of X-ACS are that it allows the automated validation of policies and can be used by processors with limited capacity such as smartcards.

1 Introduction

Although the Internet and the World-Wide Web have lead to the popularization of distributed systems in most computing disciplines, systems for access control to information still rely on centralized security administration. Centralized control has important disadvantages: (a) The control point represents a weak spot for security attacks and fault tolerance, (b) it does not facilitate the deployment of originator retained control mechanisms, (c) it reduces system performance because it introduces a bottleneck for request handling, and (d) it usually enforces homogeneous access control schemes that do not fit naturally in heterogeneous user groups and organizations.

On the other hand, distributed access control is still an open problem. Solutions proposed so far fail to provide the flexibility and manageability required. A system for distributed access control has been recently proposed based in the concept of mobile policies [1] to solve some of the limitations of RBAC [2]. This system is based in the remote execution of the access control policies. Therefore, it addresses a solution for some of the problems of centralized access control. In this proposal three important assumptions are made: (i) it is assumed that mobile policies and the associated data that they control are indivisible, (ii) that the system will guarantee that the policy is always executed and enforced before access is granted to an object and, finally, (iii) it is assumed that the policy is always executed in trusted computers. This approach is consequently limited by the requirement of executing the access control policies in trusted computers (data servers in this case) which in practice represents just a small improvement over the single-server model. Furthermore, when access to a data object is granted, this data has to be send to the client computer where it has no protection. Finally, because data and policy are compiled in a package, a change in the policy that controls a data object requires that the data-policy package is recompiled and distributed to all trusted servers.

In this paper we present ACDACS (Attribute Certificate Distributed Access Control System). ACDACS is based in extension of the mobile policy concept that allows us to execute the policies in untrusted systems. We also extend the scheme to allow that the policies are linked to the data but not integrated with it. This modification permits that policies are dynamically changed in a transparent manner. We also introduce X-ACS; a XML-based authorization language that is designed to support all the possible authorization scenarios in a simple and unambiguous way and to facilitate policy validation processes.

The rest of the paper is organized as follows. Section 2 presents the motivations. Section 3 summarizes some related work. Section 4 describes the ACDACS system. Finally, section 5 summarizes the conclusions and presents ongoing and future work.

2 Motivation

Inside distributed computing environments, such as extranets or research networks that comprise several institutions, the access policy applicable to each resource (data object, service, etc.) must be defined by the owner of the resource. The system must guarantee that the policy is enforced before access is granted to the resource. Moreover, there are many different situations where it is desirable that the owner of each resource is able to retain the control over it and to change the access policy dynamically and transparently. In these systems traditional centralized access control mechanisms do not provide the necessary functionality and flexibility. The need of a central authority and repository is not always acceptable by the institutions sharing the network. Furthermore, centralized systems are unable to provide means to guarantee that originators retain control over their information.

Several access control models have been introduced in the literature to fit different access control scenarios and requirements. Some schemes have also tried to integrate different models in a unified framework [3]. These approaches represent significant advances over traditional single-policy systems but, unfortunately, are still constrained by the underlying models and do not provide the necessary flexibility.

Role based access control is commonly accepted as the most appropriate paradigm for the implementation of access control in complex scenarios. RBAC can be considered a mature and flexible technology. Numerous authors have discussed the access properties and have presented different languages and systems that apply this paradigm [4-7]. Commercial implementations exist based in RBAC schemes.

The main problem with role based access control is that the mechanisms are built on three predefined concepts: "*user*", "*role*" and "*group*". The definition of roles and the grouping of users can facilitate management, specially in corporation information systems, because roles and groups fit naturally in the organizational structures of the companies. However, when applied to some new and more general access control scenarios, these concepts are somewhat artificial.

We believe that a more general approach is needed in order to be used in these new environments. For example, in the referred situations, groups are an artificial substitute for a more general tool: the *attribute*. In fact, groups are usually defined based in the values of some specific attributes (employer, position, …). Some attributes are even built into most of the access control models. This is the case of the *user* element; the identity is just one of the most useful attributes, but it is not necessary in all scenarios and, therefore, it should not be a built-in component of a general model.

In actual access control models, the structure of groups is defined by the security administrator and it is usually static. Although the grouping of users can suffice in many different situations, it is not flexible enough to cope with the requirements of more dynamic systems where the structure of groups can not be anticipated by the administrators of the access control system. In these scenarios new resources are incorporated to the system continuously and each resource may possibly need a different group structure and access control policy. Furthermore, the policy for a given resource may change frequently.

Finally, because the creation and maintenance of access control policies is a difficult and error prone activity, we have developed a language to express those policies and validate them to find contradictions or ambiguities.

Our work is focused in the solution of the originator-retained-control issue providing fair distributed access control management and enforcement. The basic goal is to be able to express, validate and enforce access control policies without assuming the trust in the rest of the computers of the network.

3 Related work

There are a number of research groups working on specification of access control policies. In [8] an interesting system is presented for policy based management of networks and distributed systems. The separation of the policy specification from the access control implementation has been proposed in [9]. This separation follows the "network-centric" approach of Röscheisen and Winograd [10] and allows the policy to be modified dynamically, without changing its underlying implementation [11].

Several proposals have been introduced for access control to distributed heterogeneous resources from multiple sources. The outcome of the Akenti Project [12] is an access control system designed to address the issues raised in allowing restricted access to distributed resources (information, processing or communication capabilities, or physical systems such as scientific instruments) which are controlled by multiple stakeholders. Drawbacks of the Akenti access control system are the requirement for the stakeholders to trust the rest of the servers in the network, the assumption of the existence of a supporting identity PKI (public key infrastructure) and some security vulnerabilities related to the existence of positive and negative use-conditions.

The PERMIS Project [13] objective is to set up an integrated infrastructure to solve identification and authorization problems. A specific goal is to specify the authorization policy in a language that can be both easily parsed by computers and read by the security administrators with or without software tools. Various pre-existing policy languages – for example, Ponder [8] – have been examined, but none has been found to be ideally suited. The PERMIS group concluded that XML is the most appropriate candidate for a policy specification language. However, because PERMIS system is based on the RBAC model, it shares its limitations. Moreover, the requirement of supporting a PKI is hard to fulfil and it is not necessary in many authorization scenarios.

Since its inception in 1998, XML has been used for defining specific vocabularies to represent different human endeavor. In the context of security, the eXtensible rights Markup Language (XrML) [14] and extensible Access Control Markup Language (XACML) [15] are two proposals for standard XML extensions in the fields of digital rights management and access control. While XrML can be considered a mature specification (first efforts started in 1996), it is not appropriate for our application scenarios where it is essential to keep specifications simple. XACML is a very recent (the XACML group started their work on April 16th, 2001) and promising competitor in the field of access control languages. The 0.8 version of the language was released on January 10th, 2002. Therefore, XACML is still in the development process. XACML is based in XML-Schema as is X-ACS. The main differences are that X-ACS is designed to be used by processors with limited storage and processing capabilities such as smartcards and oriented to the validation of the policies.

Also based on XML, the Security Assertion Markup Language (SAML) [16] is an assertion language and messaging protocol for describing, requesting and sending authentication and authorization data between security domains. The basic goal of SAML is to promote the interoperability between disparate security systems, providing the framework for secure e-business transactions across company boundaries.

Another interesting work is presented in [17,18], where XML is used for defining a fine-grained access control system for XML documents. This approach differs from ours in that it is completely 'server-side'. Authorizations can be specified at document or instance level (in XML documents), or alternatively at schema level (in Document Type Definition –DTDs-). Authorizations specified in a DTD are applicable to all XML documents that conform to it. Our proposal is based in a higher level schema language, the XML-Schema language [19], proposed by the World Wide Web Consortium (W3C) as the replacement for DTD, which present a XML syntax and advanced features such as inheritance, enhanced data types, etc. XML-Schema can be extended with business rules expressed in Schematron [20]. The expressive power of both languages allows the definition of advanced integrity constraints in a database-like style [21].

Other access control languages have been developed in the security community to support different access control approaches. Jajodia et al. present in [22] a logical language which allows users to specify the policy according to what access control decisions are to be made as well as the authorizations. Our work is focused is in this direction, but in this case we are interested in access control for highly dynamic systems with an important volume of heterogeneous data and multiple independent data sources and the originator-retained-control problem. We use XML along with XML-Schema to enable the definition of policies expressed by means of rules and the representation of integrity constraints to be verified.

4 The ACDACS system

Taking into account the basic objective of providing means to solve the originator-retained-control issue, the different scenarios considered and the analysis of the previous proposals, our main goals for the ACDACS distributed access control system are:

- *Originator-retained-control.* Originators should be able to retain control on the resources they own even after access is granted to users.
- *Distributed access control management.* Administrators should be able to manage the resources they control regardless of the location of that resource.
- *Distributed access control enforcement.* Access control mechanisms must be distributed in order to avoid bottlenecks in request processing.
- *Flexibility.* The system should be applicable in different scenarios.
- *Independence.* The system should not depend on underlying infrastructures or authentication systems.
- *Dynamism.* There should be a fast and secure mechanism to change policies.
- *Ease of management.* The distributed approach should not introduce complexity of management.
- *Efficiency.* Both access control management and enforcement should be efficient.
- *Security.* The distributed access control mechanism must ensure the same level of security as a centralized one can achieve. Tools to help security administrators should be provided.

4.1 ACDACS system architecture

The ACDACS system is based on the following idea: the security requirements of the processes related to the transmission and access to information are feasible if we can have a trusted software running in the client computer. Therefore, the system is based in the creation of mobile software elements that are responsible for the transport of the protected content and the enforcement of the access control.

We present an architecture that allow us to securely execute the access enforcement mechanism in untrusted systems. Two different alternatives are available for the software protection mechanism. The first one can be used without an online connection when accessing the information but requires the use of special smart cards. The second one requires the use of an online connection to some special access servers.

The architecture is based in the *SmartProt* software protection system described in [23]. This system is based in the partition of the software into functions that are executed by two collaborating processors. One of those processors has to be a trusted computing device that enforces the correct execution of the functions and avoids that these functions are identified or reverse engineered.

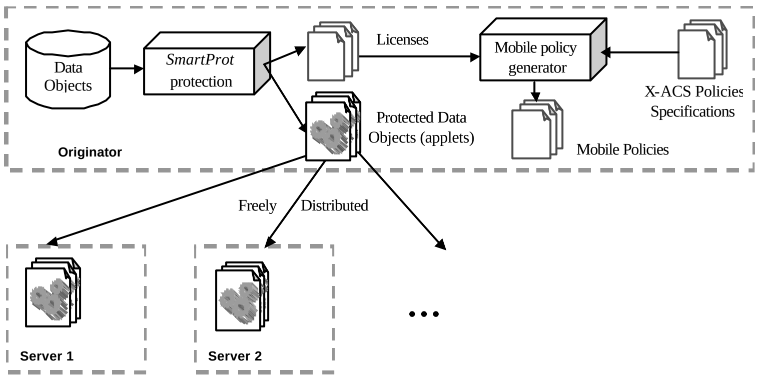

Fig. 1. The ACDACS Architecture

Figure 1 shows the architecture of the system. The unprotected data objects in the originator computer are transformed into PDOs (protected data objects), Java applets that protect the data and enforce the access control mechanism. Policies are not included in the PDO, instead, each PDO is linked to the applicable policy by a mobile policy that is produced specifically for the client when requested. PDOs can be freely distributed to untrusted servers.

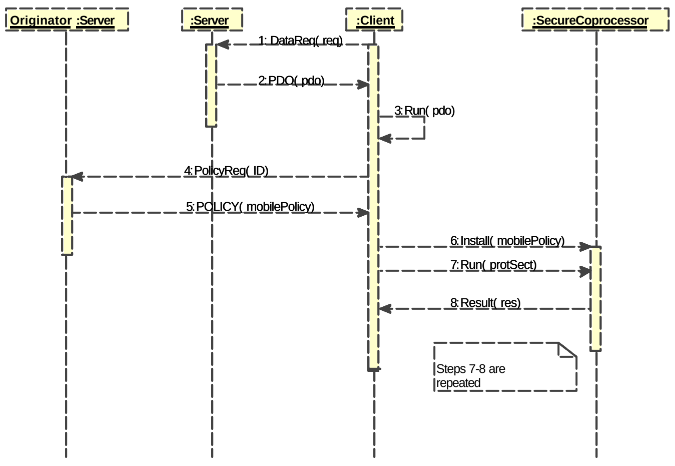

Fig. 2. Functioning of the ACDACS system

Figure 2 shows the process to access the information. When the client requests some data object from a server it receives the PDO containing it. This PDO runs in the client computer. Before the PDO can execute the protected sections of its code it has to retrieve the corresponding mobile policy (which includes the license that allows the decryption and execution of the protected sections of the PDO). To do this it sends a request containing the certificate of the public key of the secure coprocessor (the smart card or the access server). In case the server from where the PDO was retrieved is the originator of the PDO, it produces the mobile policy for that PDO. Otherwise it just forwards this request to the PDO originator.

In scenarios where the number of attributes and attribute certification authorities, also known as SOAs (source of authorizations), are high it might be desirable to avoid that clients have to verify all the certificates directly. In this case a temporary authorization mechanism is used to map several attribute certificates from different SOAs to a single temporary authorization. This temporary authorization is a special attribute certificate signed by one SOA (probably the originator). This simplifies the verification that the PDOs perform and is specially useful when the client is accessing a large number of PDOs from the same originator.

One important constraint to the free distribution of protected information in our system is that owners of the information must be able to dynamically change the access control policy. For this requirement to be fulfilled we have to separate the policy from the PDO. Policies are retrieved from the originator during the execution of the protected software and are enforced by this software. The reasons to require the request of the mobile policy at access time and from the originator are that it allows a high degree of flexibility, and gives the originator more control over the application of the policies. Nevertheless, originators can define certain validity constraints for each policy (based on time, number of accesses, etc. depending on the smartcard features). Therefore policies can be cached by clients and used directly while they are still valid. Also the generation of the mobile policy is a reasonably fast process while the generation of PDOs is slower. Furthermore, PDOs are much more stable than policies. Finally, opposed to PDOs, each mobile policy is specific for a smart card (or access server).

Policies are specified using X-ACS and are later translated into a more compact form to be included in the mobile policy. The link between PDO and the corresponding mobile policy is established by cryptographic means. The structure of the mobile policy is defined as follows:

$$\text{MP} ::= \text{Policy, Encrypt}_{\text{CardPublicKey}}(\text{PDOkey, validity, H(Policy)})$$

where *Policy* is the compact representation of the X-ACS policy, *CardPublicKey* is the public key of the smart card that will access the PDO, *PDOkey* is the random symmetric key used to encrypt the protected sections of the PDO, *validity* represents the limits of use of the MP and *H* is a collision-resistant one way hash function.

The mobile policy includes the key required by the smart card to decrypt and run the protected sections of the PDO. This key is encrypted and will only be in the clear inside the right smart card. As the PDO key is only known by the originator it is impossible for dishonest users to alter mobile policies or produce false ones.

We will now describe the main building blocks of ACDACS: (i) a security infrastructure for software protection and (ii) a policy specification and validation language.

4.2 Software Protection

One of the main security problems in information access control systems is the difficulty for the owner of the information to retain the control over it after it is accessed by users. The main reason for this is that most security mechanisms are designed to protect the information while in transit over the network. The result is the information being defenceless when it (securely) arrives to the client computer. To deal with this situation, persistent protection mechanisms must be used.

The ability to protect software that runs in the client computer in order to guarantee that it performs the function that it was intended to, opens a way to solve the previous problem. Our solution for the originator-retained-control problem is based in the protection of the mobile software that we use to convey the information. In this section we will describe the mechanisms used to protect the mobile software in the ACDACS system. In the following description we use smart cards as secure coprocessors although, as mentioned, special online access servers can also be used as secure coprocessors for this scheme.

The SmartProt system includes three actors: the information provider, the card manufacturer and the client (who possess a smart card). The card manufacturer certifies the public keys of the smart cards. SmartProt requires smart cards that have cryptographic capabilities, contain a key pair generated inside the card and ensure that the private key never leave the card. Cards also contain the public key of the card manufacturer and some support software. In particular, cards contain an interpreter for the protected code sections, a license manager, a runtime manager and sometimes also an electronic purse. Each protected software application runs in a sandbox isolated from others. The contents of the memory used by one application remains between calls to the card.

Information providers (originators) decide which card manufacturers (or access servers entities) to trust and is assumed that they have access to digital certificates of the public keys of these manufacturers. It is presumed that there will be a reduced number of these manufacturers.

The software contains some protected sections that must be executed by the card. a license stating conditions like validity, etc. is needed to run the protected software.

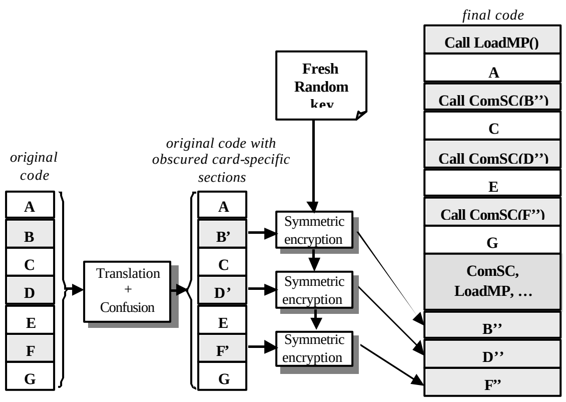

Fig. 3. Code transform (production phase).

The production phase is schematized in figure 3. The first step of this phase consists in the translation of some specific sections of the original application code by functionally equivalent sections of card-specific code. The translation process also identifies the dependencies between these protected sections, reorganizes the code and introduces fake code to confuse the attacker. These sections are then encrypted with a unique randomly produced key using a symmetric cryptosystem. The last step substitutes the original code sections by calls to a function that transmits the respective equivalent protected sections, including code and data, to the card. Some additional support functions are also included. The protected mobile software application generated in the production phase can be distributed and copied freely. In the case of the ACDACS system, the protected application (PDO) is a Java applet responsible for the transport of the information. Therefore, the protected mobile software includes the information to be accessed (which is encrypted), the access control enforcement mechanism and a cryptographic link to the access policy. The production phase is independent of the client card and will be performed just once for each piece of mobile software. In the original software protection system, a license (specifically created for the smart card of the user) is required to run the protected software. In the ACDACS system the license includes the access policy and is called Mobile Policy (MP).

In the authorization phase, the new MP is produced linking the policy and the PDO. The MP contains validity constraints that are set by the security administrator according to the volatility of the policies. In case a user accessing a PDO already has a non-expired MP for it, the existing MP can be used, otherwise a new MP has to be requested and produced. The MP is obtained and loaded in the card as part of the PDO. When the MP is received by the client smart card it is decrypted, verified and stored inside the card until it expires or the user explicitly decides to extract it. Once the MP is correctly installed in the card the protected program can be executed, which requires the cooperation of the card containing the MP. The protected sections of the software do not reside in the cards; instead, during the execution of the protected program, these sections are transmitted dynamically as necessary to the card where they are decrypted using the installed MP and executed. When finished, the card may send back some results. Some other partial results will be kept in the card in order to obtain a better protection against function analysis and other attacks.

To summarize, SmartProt allows a single card to be used to protect many applications, permits a high degree of complexity in the protected sections, allows the card to execute any number of those sections and enables the distribution of the software through Internet because none of the components of a protected application needs to be preloaded in the hardware (card). The definition of the license (MP) structure permits a high degree of flexibility. Furthermore, as each application has its own key, we can manage them individually, which is not possible in other software protection proposals where the protected sections of all applications are protected using the same key (usually the protected processor key). In this scheme, because the protected sections are encrypted using a symmetric key that is kept inside the cards, and is therefore known only by the software producer, it is impossible for a dishonest user to produce false sections.

4.3 Authorization language

The XML data model can represent semantic information through descriptive tags. Complemented by related technologies certain types of database-like schemes can also be used [21].

XML DTD is the de facto standard XML schema language but has limited capabilities compared to other schema languages [24], such as its successor XML-Schema. This language presents a rich set of data types, allowing user-defined data types, mechanisms such as inheritance, etc. Moreover, XML-Schema uses XML syntax, enabling the use of XML tools. XML-Schema is the proposal of the W3C for a more expressive language to replace DTDs. Our work is based on this proposal as it allows us to represent the semantics of the policies. Although this schema language is very powerful, there are some constraints that it can not express. For example, "the value of one <attribute_value> element in a document is greater than the value of the corresponding <attribute_value> element in another document", or "if the value of the <Rights> is 'Update' then the value of the <Actions> element should be 'Notify'" (co-occurrence constraints).

Schematron is a rule-based schema language well suited to express this kind of constraints embedded within <appinfo> elements in XML-Schema documents. In fact, Schematron has its strengths where XML-Schema has its weaknesses (co-occurrence constraints) and its weaknesses where XML-Schema has its strengths (structure and data types). The Schematron rules uses the XPath language with the various extensions provided by XSLT. Nevertheless, in time critical applications, the overhead of processing the embedded Schematron rules may be too long. It should also be noted that the Schematron rules can only be applied on specific elements in the XML instance document. It is not possible to apply Schematron rules to complex type definitions in XML-Schema. Regrettably, existing implementations do not provide all the features required in our application.

```
<?xml version="1.0"?>
<xsd:schema  targetNamespace="http://www.lcc.uma.es/ecWeb2002"
             xmlns="http://www.lcc.uma.es/ecWeb2002"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             elementFormDefault="qualified">
  <xsd:simpleType name="policy_ID_type">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z]{3}-[0-9]{3}"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="Rights_Type">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="read"/>
      <xsd:enumeration value="update"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="Actions_Type">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Notify"/>
      <xsd:enumeration value="OnLine_Permission"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="policy">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element    name="parameter" type="xsd:string"
                        minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="access_Rules">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="access_Rule" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="policy_Description" type="xsd:string" use="optional"/>
      <xsd:attribute name="policy_ID" type="policy_ID_type" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="access_Rule">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="attribute_Set" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="Right" type="Rights_Type" use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="attribute_Set">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="attribute" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="Action" type="Actions_Type" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="attribute">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="attribute_name"/>
        <xsd:element name="attribute_value"/>
        <xsd:element name="SOA_ID" type="xsd:string" nillable="false"/>
      </xsd:sequence>
      <xsd:attribute name="attributeID" type="xsd:ID" use="optional"/>
      <xsd:attribute name="attributeDescription" type="xsd:string" use="optional"/>
  <!-- Each (attribute_name, attribute_value) is certified by a signer - SOA_ID - -->
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Fig. 4. PolicyTemplate.xsd

Figure 4 shows the schema of X-ACS policies. This XML-Schema template facilitates the creation of X-ACS policies, allowing automatic syntactic validation of X-ACS policies. One of the most important constraints imposed to the design of our language is that X-ACS policies must be evaluated by processors with a limited storage and processing capability such as smartcards. For this reason, other related languages are not well suited for the ACDACS system.

In our language, a policy consists of a set of `access_Rule` elements. Each one of these elements defines all the combinations of attribute certificates that allow the user to gain the access established by the `Right` attribute. Therefore, it is composed as a series of `attribute_Set` required to gain access and the `Right` obtained over the data in case access is granted. Each `attribute_Set` defines a particular attribute certificate combination associated with an optional `Action` (that has to be performed before access is granted).

Attribute certificates will be used to provide evidence of the possession of each attribute. Therefore, attribute certificates are described stating their name (`attribute_name`), value (`attribute_value`) and the signer of the certificate (`SOA_ID`).

Figure 5 shows an instance of the PolicyTemplate.xsd schema containing an example policy.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<policy   xmlns="http://www.lcc.uma.es/ecWeb2002"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.lcc.uma.es/ecWeb2002 PolicyTemplate.xsd"
          policy_ID="POL-123"
          policy_Description= "GRANT Read Access TO users possessing a
                              temporal authorization WITH Notification
                              and GRANT Read and Update Access TO Professors">
 <access_Rules>
    <access_Rule Right="read">
       <attribute_Set>
          <attribute>
             <attribute_name>Position</attribute_name>
             <attribute_value>Professor</attribute_value>
             <SOA_ID>LCC_ADM</SOA_ID>
          </attribute>
       </attribute_Set>
       <attribute_Set Action="Notify">
          <attribute>
             <attribute_name>Tmp_Auth</attribute_name>
             <attribute_value>P123</attribute_value>
             <SOA_ID>UMA_ETSII</SOA_ID>
          </attribute>
       </attribute_Set>
    </access_Rule>
    <access_Rule Right="update">
       <attribute_Set>
          <attribute>
             <attribute_name>Position</attribute_name>
             <attribute_value>Professor</attribute_value>
             <SOA_ID>LCC_ADM</SOA_ID>
          </attribute>
       </attribute_Set>
    </access_Rule>
 </access_Rules>
</policy>
```

Fig. 5. ExamplePolicy.xml

Suppose our administrator wants to grant authorization to access the marks of a course to professors and deny it to students. Without a careful analysis, the administrator states the following X-ACS policy:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<policy xmlns="http://www.lcc.uma.es/ecWeb2002"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.lcc.uma.es/ecWeb2002 PolicyTemplate.xsd"
        policy_Description="GRANT write_access TO marks IF Position='Professor' ">
  <access_Rules>
     <access_Rule Right="update">
        <attribute_Set>
          <attribute>
             <attribute_name>Position</attribute_name>
             <attribute_value>Professor</attribute_value>
             <SOA_ID>LCC_ADM</SOA_ID>
          </attribute>
        </attribute_Set>
     </access_Rule>
  </access_Rules>
</policy>
```

Fig. 6. WrongPolicy.xml

The problem with this policy is that, in some institutions, it is possible for a professor to be registered as student in courses other than those he teaches. In such a case, students who are also professors would get access to their own marks.

X-ACS policies are verified in several ways. Policies are verified syntactically using XML-Schema. Semantic verification is made possible by the use of a specific

Policy Validator that uses the SAX API to parse the document validating it. Finally, policies can be verified to check their validity in the context where they will be applied. Policy context verification is based on the definition of a set of global rules (also expressed using X-ACS). This set of rules establishes a series of facts about the environment of the system (for example, in a basket team, global rules could state that the coach may also be player). This semantic information allows the detection of possible inconsistencies in the declared policy. The administrator can define the test cases to be checked.

With the aid of the context validation the administrator could have detected the error in the previous example. The rule should have been stated as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<policy xmlns="http://www.lcc.uma.es/ecWeb2002"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.lcc.uma.es/ecWeb2002 PolicyTemplate.xsd"
     policy_Description="GRANT write_access TO Target.marks IF
             Position='Professor' AND Teaches=Target">
   <parameter>Target</parameter>
   <access_Rules>
     <access_Rule Right="update">
        <attribute_Set>
          <attribute>
             <attribute_name>Position</attribute_name>
             <attribute_value>Professor</attribute_value>
             <SOA_ID>LCC_ADM</SOA_ID>
          </attribute>
          <attribute>
             <attribute_name>Teaches</attribute_name>
             <attribute_value>*Target</attribute_value>
             <SOA_ID>LCC_ADM</SOA_ID>
          </attribute>
        </attribute_Set>
     </access_Rule>
   </access_Rules>
</policy>
```

Fig. 7. RightPolicy.xml

The Target attribute is interpreted by the Validator as a parameter that has to be instantiated by the test case. The global rules established about the context in the X-ACS schema, enables the detection of semantically incomplete policies. The Policy Validator makes the verification of the policy in an automatic way.

5 Conclusions and future work

We have presented the ACDACS distributed access control system. The ACDACS approach solves the originator-retained-control problem. We have described the underlying mechanisms that make possible this system: the SmartProt software protection scheme and the X-ACS language and tools. ACDACS is flexible, can be applied regardless of the attribute certification scheme, implements distributed access control management and enforcement mechanisms, does not depend on underlying infrastructures or authentication systems, allows the dynamic modification of policies in a transparent and efficient way and is secure.

We have functional implementations of the software protection mechanism and the PDO (applet) generator. Currently, PDO applets are not signed and, therefore, the connection to the originator is done using a proxy (Oracle Connection Manager) on the servers. We plan to use signed applets in future versions.

About X-ACS we have developed the basic XML-Schema specification and the Policy Validator. Ongoing work is focused in the implementation of some of the components of the system such as the policy writer assistant. Because the administrator of the access control system has to specify what attributes from what SOAs are required to access each data object we are working in the definition of a generic mechanism to allow access control administrators to gain knowledge about the attributes certified by each SOA. This mechanism is based in tools to allow SOAs to define XML-Schemas for the attributes they certified. This XML-Schemas will be accessed by the access control administrators and used by their policy creation and validation tools. We are also interested in the application of the system to other scenarios such as digital libraries and pay-per-view.

Acknowledgements

The authors want to acknowledge the authors of [1], and specially, Amgad Fayad for their help and interesting comments about the first drafts of this work.

The work of Javier López was partially supported by the E.U. through FEDER project 1-FD97-0850. The work of Antonio Maña was partially supported by the E.U. through FEDER project 1-FD97-1269-C02-02. The work of Mariemma I. Yagüe was partially supported by the Spanish Ministry of Science and Technology through project CICYT TIC99-1083-C02-01.

References

1. Fayad, A. and S. Jajodia, *Going Beyond MAC and DAC Using Mobile Policies*. In Proceedings of IFIP SEC'01. Kluwer Academic Publishers. 2001.
2. McCollum, C.J.; Messing, J.R.; Notargiacomo, L. *Beyond the pale of MAC and DAC - Defining new forms of access control*. Proceedings of the IEEE Symposium on Security and Privacy, pp. 190-200. 1990.
3. Jajodia, S.; Samarati, P.; Sapino, M.L.; Subrahmanian, V.S. *Flexible support for multiple access control policies*. ACM Transactions on Database Systems, 2000.
4. Baldwin, R. W. *Naming and Grouping Privileges to Simplify Security Management in Large Database*. Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy, pp. 61-70, Oakland, CA, April 1990.
5. Sandhu, R.S., E.J. Coyne, H.L. Feinstein, and C.E. Youman, *Role-Based Access Control Models*. IEEE Computer, 1996. 29(2): pp. 38-47.
6. Osborn, S.; Sandhu, R.; Munawer, Q. *Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies.* ACM Transactions on Information and Sy stem Security, 3(2) pp. 85-106. 2000.
7. Sandhu, R.; Ferraiolo, D.; Kuhn R. *The NIST Model for Role-Based Access Control: Towards a Unified Standard*. Proccedings of the fifth ACM Workshop on Role-based Access Control. pp. 47-63. 2000.
8. Damianou, N., Dulay, N., Lupu, E. and M. Sloman. *The Ponder Policy Specification Language*. In Proceedings of Policy Worshop 2001. 2001.
9. Wedde, H.F. and M. Lischka. *Modular Authorization.* In Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT). 2001.

10. Röscheisen, M.; Winograd. T. *A Network-Centric Design for Relationship-based Security and Access Control*. Journal of Computer Security, Special Issue on Security in the World-Wide Web. 1997.
11. Sloman, M.S. *Policy Driven Management for Distributed Systems*. Journal of Network and Systems Management, 2(4) pp. 333-360. 1994.
12. Thompson, M., et al., *Certificate-based Access Control for Widely Distributed Resources*. Proceedings of the Eighth USENIX Security Symposium. pp. 215-227. 1999.
13. Chadwick, D. W. *An X.509 Role-based Privilege Management Infrastructure*. Business Briefing. Global Infosecurity 2002. http://www.permis.org/
14. ContentGuard, Inc. *eXtensible Rights Markup Language, XrML 2.0*. 2001. http://www.xrml.org
15. Organization for the Advancement of Structured Information Standards. *eXtensible Access Control Markup Language*. http://www.oasis-open.org/committees/xacml/
16. Organization for the Advancement of Structured Information Standards. *SAML 1.0 Specification Set*. http://www.oasis-open.org/committees/security/
17. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S. and P. Samarati. *Securing XML Documents*. In Proc. of the 2000 International Conference on Extending Database Technology (EDBT2000), Konstanz, Germany, March2000.
18. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S. and Samarati, P. *A fine-grained access control system for XML* documents. In ACM Transactions on Information and System Security (TISSEC), to appear.
19. W3C. *XML-Schema*. http://www.w3.org/XML/Schema
20. Jelliffe R. *Schematron*. http://www.ascc.net/xml/resource/schematron/schematron.html
21. Yagüe, M.I., Aldana, J.F. and C.A. Gómez. *Integrity issues in the Web*. In (Doorn, J. and L. Rivero, eds.) "Database Integrity: Challenges and Solutions". ISBN: 1-930708-38-6. 2002.
22. Jajodia, S., Samarati, P. Subrahmanian, V.S. *A Logical Language for Expressing Authorizations*. In Proceedings of IEEE Symposium on Security and Privacy. pp.31-42. 1997.
23. Maña, A. and E. Pimentel, *An Efficient Software Protection Scheme*. In Proceedings of IFIP SEC'01. Kluwer Academic Publishers. 2001.
24. Lee, D. and W.W. Chu. *Comparative Analysis of Six XML-Schema Languages*. SIGMOD Record, Vol. 29, No. 3, pp. 76-87. 2000.