Advanced Policies Management for the Support of the Administrative Delegation in Federated Systems

Manuel Gil Pérez, Gabriel López, and Antonio F. Gómez Skarmeta
Departamento de Ingeniería de la Información y las Comunicaciones
University of Murcia, Spain
Email: {mgilperez,gabilm,skarmeta}@um.es

Aljosa Pasic
Atos Origin
Albarracin 25, 28037 Madrid
Email: aljosa.pasic@atosresearch.eu

***Abstract*—Current identity management systems are experiencing an increasing workload of their administrators in the management of the system policies, mainly derived from the sheer amount of policies they have to create and maintain. This problem is even more relevant in federated environments, where roaming users force them to authenticate and authorize people coming from other institutions. In this context, it is increasingly necessary to adopt new advanced policies for the *administrative delegation*, which allow balancing this workload among several delegates who will in turn have a much wider knowledge in the application area where these policies will be applied. In this paper, we present an infrastructure that manages the entire life cycle of the administrative delegation policies in federated environments, as well as a way for reducing the complexity in their management for some scenarios, especially on those where the delegates do not have to be experts in the subject area. These delegates will only have to fill in a simple template, which is automatically generated by our infrastructure. Finally, the applicability of the proposed infrastructure is measured with some performance results.**

***Keywords*-administrative delegation, authorization policies, identity federation, access control.**

I. INTRODUCTION

This paper is an extended and revised version of the conference paper entitled "Advanced Policies for the Administrative Delegation in Federated Environments" [1]. It contains a more comprehensive and detailed explanation of the proposed infrastructure with delegation support, as well as a new section with performance measurements to demonstrate and assess the applicability of the herein introduced prototype in real identity management systems.

Mobility of users among institutions has become more and more common in recent years. For example, the Erasmus Programme [2] has promoted the academic mobility of higher education students and teachers within the European Union. Since the Bologna accords in 1999 [3], and the creation of the European Higher Education Area [4], it is expected this mobility will be increased over time.

On the other hand, we are also currently undergoing the emergence of federated identity systems with the aim of sharing resources among different autonomous institutions. Important examples of these systems are the establishment of academic federations worldwide, such as *eduroam* [5], HAKA [6], or SWITCH [7].

In these scenarios, access control policies are used to manage the access of end users to services and resources offered by an institution. However, as the number of members of an institution increases, new institutions join to the same federation or the relationships among them change, the management of these policies becomes more and more complex. This is due mainly to the great amount of policies to manage, either access control policies, privacy policies or validation policies based on *Levels of Assurance* (LoA) [8], among others.

To reduce the complexity in the management of these policies the *administrative delegation* allows system administrators to delegate some privileges to others, named *delegates*, with the aim of making part of their work by managing a subset of the system policies [9]. In this way, not only is the management of the system policies distributed to other people, but also they are being delegated to people who have better knowledge on the application area upon which these policies will be used.

As an example, the system administrator of an institution may delegate in the head of a department to specify which of the members of her department can access the network. This delegate will be also able to establish certain constraints under which her employees can do it, e.g., they will be only able to access the network in an specific time interval.

This new sort of policies supposes a new value-added service to the current policy-driven systems, either federated or not, although its use also introduces some drawbacks that have to be treated adequately:

- The number of policies to manage increases dramatically. System administrators will have to manage both the policies that already existed (access control policies or privacy policies, among others) and this new kind of policies to control the administrative delegation. It will introduce a new way of controlling which users can create new policies (administrative policies).
- Delegates are usually users with no knowledge in policy management, access control languages, etc. Thus, we should make it easier for those people the generation

and management of this new kind of policies.

As seen, although the workload of administrators is reduced, or distributed considerably among several delegates, the policy management (including the administrative ones) will also be more complex. Thus, the definition of these new policies for the administrative delegation is not enough for its deployment in real environments, but it is also necessary to define an infrastructure that can manage them and help delegates to do their new tasks.

As a solution to these problems, we will include a set of new components to existing federated identity systems to manage the complete life cycle of the administrative delegation policies. Moreover, we will also define a new mechanism for the generation of templates that helps delegates to carry out this new task in a simple and intuitive fashion. These templates will be automatically generated by our infrastructure from the administrative delegation policies created by the system administrators.

The remainder of this paper is organized as follows. Section II describes an example scenario that is used to motivate this research work. The access control language used in this proposal and its extension to enable the use of the administrative delegation are shown in Section III. Section IV describes in detail the infrastructure providing administrative delegation, whereas Section V presents the automatic generation of policies and templates for helping delegates to do their tasks. Section VI illustrates some performance measurements to assess the applicability of the prototype in real identity management systems. Then, Section VII presents the main related work and, finally, Section VIII remarks the main conclusions and future work.

II. USE CASE

As an application example of this new kind of policies let us suppose a scenario where an institution is going to host an international project meeting, in which members from other institutions need Internet access. Then, the host institution will provide such a connection with certain *Quality of Service* (QoS) assurances. All participants, coming from various institutions, belong to the same identity federation.

Figure 1 depicts this situation, where two participants coming from different institutions (Bob from *Institution A* and Carl from *Institution B*), but all belonging to the same identity federation than the host institution, want to get a connection to the Internet.

The administrative delegation can be used in this scenario to assign the responsibility of managing the access control properties to a user more closely related to the mentioned scenario. For instance, the person who is organizing the meeting; she knows all the necessary information, such as the identity of the audience or the meeting schedule. In this way, the host institution's administrator will delegate to the meeting organizer, i.e., the delegate, the definition on which participants will have access to the network, as well as the schedule upon which they can do it; information the meeting organizer perfectly knows.

For this example, let us suppose Alice is the meeting organizer, or delegate, who will define all access control policies for the meeting participants. Then, the system administrator, besides establishing this delegation of privileges to Alice, will only have to define the QoS assurances the system should apply. This information are network parameters too much technical the delegate does not need to know.

We can prove with this scenario the use of the administrative delegation, where it prevents the administrator has to create access control policies on a group of people he does neither know nor has information with respect to the requirements each one needs.

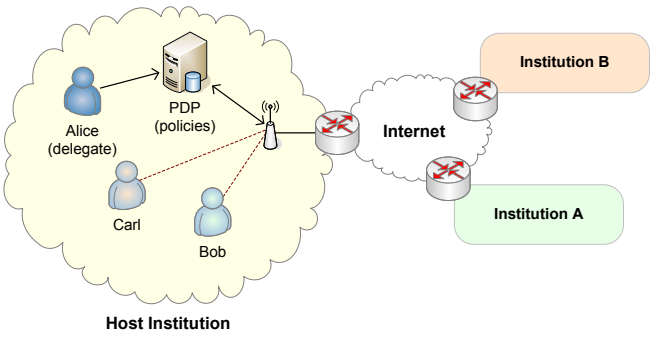

Figure 1. Example of an international project meeting

Another interesting example related to the administrative delegation can be found in multi-stakeholder scenarios such as outsourcing [10]. Some business processes are relying on IT systems of contracted service providers, the complexity of evidence collection is very high. The business processes that are subject to compliance are often scattered across multiple business units in a variety of unorganized and unmanaged systems, so design and implementation of internal control processes are not an easy task.

In this context, the MASTER EU-IST project [11] focuses, among other research topics, on automation of evidence collection. This is done with the support of MASTER operational infrastructure that relies on a set of indicators, measurable and observable properties derived from system events and configuration policies, which are set up at various levels in order to ensure trustworthy control process. What makes MASTER especially interesting is the possibility to separate evidence collection from posterior evidence correlation or compliance assessment.

In the emerging service delivery models, e.g., cloud computing, there is often an outsourcing chain where customers contract a service provider in order to do *Business Process Outsourcing* (BPO). In its turn, these service providers can store data in cloud or use some other cloud computing resources, which is not necessarily belonging to the same identity federation. As a consequence, the service provider

may not be able to offer all the required evidences to the customer. Thus, the customer is constrained to the offered granularity and semantics of the provided events, as well as monitoring and enforcement capabilities of the service provider.

Another constraint for the administration of evidence-collection process in multi-outsourcing context is the perceived lack of trust. Customers might believe that events or related evidences provided by service providers (or their subcontractors) are not authentic.

The administrative delegation mechanism proposed in this paper offers novel ways to increase the trust level. On the one hand, the trust level could be increased by applying more distributed monitoring and configuration policy rules, as well as fine grained access control policy to MASTER components and policies. Increased number of policies and decoupling of components responsible for signaling and monitoring of events (both needed for evidence collection) increases also complexity and administrative risks, which in its turn can be also addressed with administrative delegation where internal control process owner may delegate or specify who can have access to which evidence collection components. This way, each configuration of the MASTER operational infrastructure corresponds to a set of component access control policies, managed by the person who will have more knowledge about her employees than the client or service provider control process owner.

III. Administrative delegation in XACML

As mentioned before, we have identified the administrative delegation as a good alternative to manage the policies of complex systems. This section defines the mechanisms included in the eXtensible Access Control Markup Language (XACML) [12], a standard XML-based access control language, to allow the use of this new feature.

This proposal was defined to represent access control policies in a standard way. It 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 (a way to express queries about whether a particular access should be granted) and their related responses.

The administrative delegation in XACML relies on the idea that a person authorized to delegate certain privileges does not need to use them, and vice versa; that is, that a person owns rights to exercise a privilege does not imply that she can delegate it to others.

As the XACML delegation profile [13] specifies, the purpose of the delegation model is to make it possible to express permissions about the right to issue policies and to verify the issued policies against these permissions. This profile defines a new XML element, named *PolicyIssuer*, to indicate who has issued a specific policy. Through this element, the system can identify and verify whether the policy issuer is valid to delegate the given privilege before being applied. Thus, the authority of the issuer needs to be verified in order to consider this policy as valid. When a policy does not include the *PolicyIssuer* element, the policy is considered trusted and, therefore, it is valid.

By including this new element, the administrative delegation allows the creation of *delegation chains*, where a certain privilege can be delegated from one person to another. For example, in the use case shown in Section II, the meeting organizer (Alice) could in turn delegate in Bob the generation of the access control policy for Carl, thus building a delegation chain of three policies: an administrative policy, created by the administrator, that delegates in Alice the privilege of accessing to the network; another administrative policy, created by Alice, that delegates in Bob the same action and resource; and the access control policy, created by Bob, that finally grants access to the network to Carl.

In a schematic way, this delegation chain would be as follows (each arrow represents a delegation policy):

$$Administrator \rightarrow Alice \rightarrow Bob \rightarrow Carl$$

Each delegation policy can specify another XML attribute, named *MaxDelegationDepth*, to limit the depth of delegation that is authorized by such a policy. In this example, if the system administrator decides to restrict the delegation chain to only one person, by defining *MaxDelegationDepth="1"*, Alice will not be able to delegate her privileges to anyone else. Otherwise, the delegation chain will not be trusted and Carl will not have access to the network.

Thus, besides the traditional policies for managing the access to the resources (*access policies*), and the requests the *Policy Decision Point* (PDP) receives, which should be resolved based on these policies (*access requests*), this new specification also defines a new set of XACML policies to validate the issuer of another policy (*administrative policies*). This set of policies will be consulted to the PDP by means of another sort of requests (*administrative requests*).

As an example, Figure 2 depicts the delegation model in XACML for the use case presented in Section II. As seen in this figure, once the PDP receives an access request, and the policy used for its evaluation includes the *PolicyIssuer* element, as the one shown in Figure 2a, it is necessary to carry out an administrative request to verify whether the policy issuer is trustworthy, and she has the expected permissions for such a delegation. This administrative request is built from the previous access request, which will include the attributes of both users (Alice and Bob) gathered from their home institutions.

Finally, the administrative request is evaluated using the corresponding administrative policy, as the one shown in Figure 2b. It is worth noting that if this administrative policy in turn contains the *PolicyIssuer* element, the above process must be repeated until either reaching a trusted policy or

exceeding the maximum delegation depth defined in the delegation chain.

In this scenario, the policy issuer (whose identifier is *Alice*) authorizes, through the access policy, to another person with identifier *Bob* (subject) to *Access* (action) the *Network* (resource). It is also included a time condition indicating the period of time within which the policy is valid, i.e., the meeting schedule. The previous access policy is conditioned to the issuer is recognized as valid for carrying out such a delegation. To this end, the administrative policy of Figure 2b permits that the access to the network can be managed by those delegates who have the attribute *schacUserStatus* with the *meeting:set* value, and the additional condition that users requesting the access own the attribute *schacPersonalPosition* with the *Researcher* value [14]. Thus, if Alice and Bob comply with these attributes, both policies will be evaluated as valid and, finally, Bob will have the requested network connection. In this case, an obligation is also attached, a QoS requirement, which has to be enforced in the system by the *Policy Enforcement Point* (PEP).

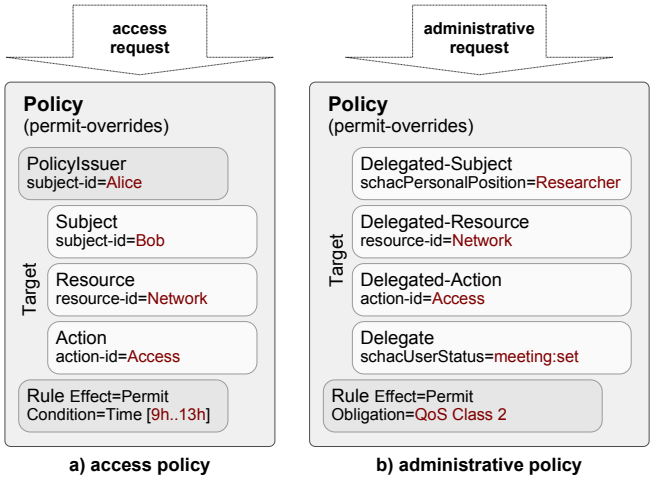

Figure 2. Administrative delegation in XACML

As has been seen in this section, the XACML delegation profile defines the syntax for the new elements that provide administrative delegation support. However, it is necessary an infrastructure that makes easy the use of this new sort of policies, as well as some mechanisms to help, especially to the delegates, in the management of these policies in an easy and intuitive fashion. Both of them are described in detail in the following sections.

Furthermore, XACML is only focused on intra-domain systems without providing any support to federated environments. Therefore, we must also take into account the identity management in our administrative delegation architecture for inter-domain systems. In this case, the system administrator will be also able to delegate part of her duties not only to people of his very institution, but also he will be able to do it to outsiders as long as they belong to the same federation his institution.

IV. Infrastructure with delegation support

Once the application area and the language to express delegation have been presented, this section describes the proposed infrastructure that makes use of delegation policies.

A. System requirements

The needs and the minimum requirements that any administrative delegation system should comply with are summarized as follows:

1) The institution wants to offer an administrative delegation service is required to provide a secure repository where to store the administrative policies.
2) The administrative policies have to be published and stored in an internal repository through secure channels that provide features of confidentiality, authentication and integrity. Moreover, it is also advisable that these secure channels provide *non-repudiation* features. This last property will avoid delegates can refuse the creation and modification of those policies for which they are responsible.
3) Efficient authentication methods are required. Only authorized people can both access the secure repository and exclusively create or modify the policies to which they have permissions. In this case, administrators will have a total access and control to all stored policies, while delegates will be only able to create and modify those policies to which the administrator has provided access, creation and modification rights.
4) The secure repository with the administrative policies has to be accessible by the service that takes the authorization decisions, i.e., the PDP, according to such policies.
5) The delegate should be capable of defining policies without having technical knowledge in managing access control policies.

B. Federation environment

An identity federation system is composed by several institutions in which a set of common services are offered, such as the authentication and authorization of the federation members. In this way, when a roaming user moves from her *home institution* to another, the *visited institution*, the later can authenticate her through the federation. In some scenarios, this authentication phase is carried out remotely by the user's home institution.

For example, *eduroam* is one of the largest networks for roaming worldwide, oriented to institutions involved in research and education [5][15]. This network allows the mobility of users across over 40 countries throughout three continents, including China, Australia and Canada.

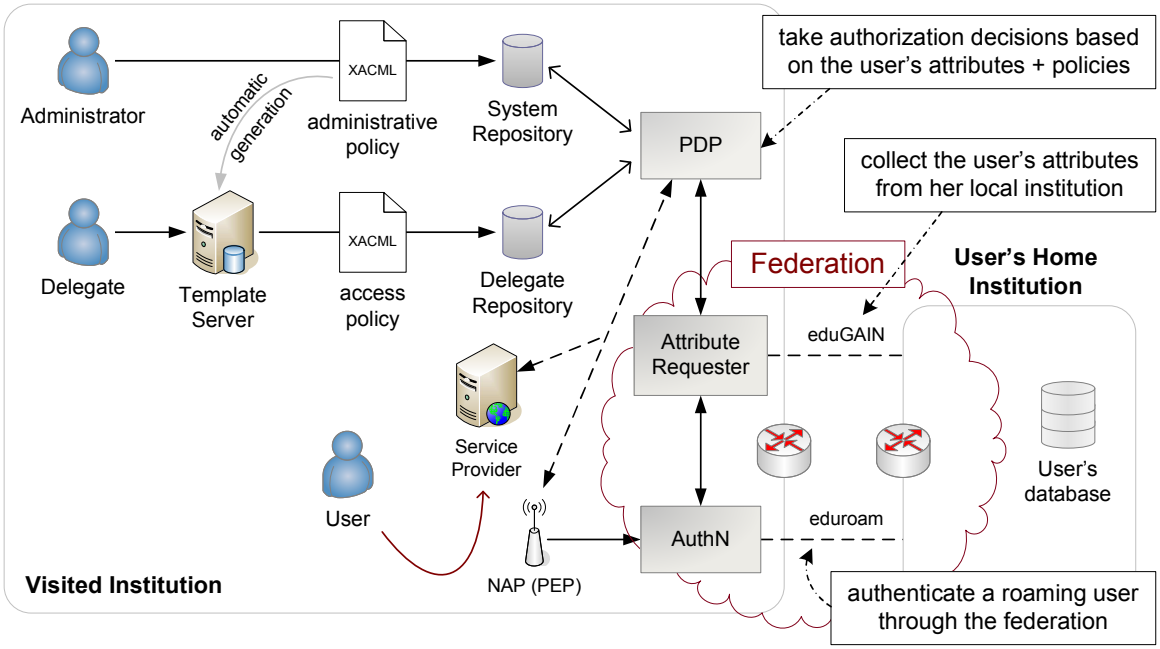

Figure 3. Administrative delegation architecture

The *eduroam* network is based on a RADIUS hierarchy that redirects the user's authentication request to the appropriate home RADIUS server. During this phase, the user receives some kind of handler to identify her in the federation, thereby providing a way of preserving her privacy in the federation. Later on, if the visited institution needs some user's attributes, to take a local authorization decision by the PDP, an attribute request can be made through the federation using the previous handler.

To that end, each institution of the federation must host two different entities (see Figure 3): an *AuthN* module to authenticate a remote user through the federation, returning her a handler (opaque identifier); and an *Attribute Requester* module that returns the attributes associated to the user identified by the previous handler. In our case, the authentication is based on *eduroam*, whereas the attributes request is carried out through *eduGAIN* [16][17]. Finally, authorization decisions are locally taken in the visited institution, according to these user's attributes and the system policies defined by the institution.

This identity federation system is a good testing environment to make use of delegation policies in a feasible way.

C. Components of the architecture

The two services previously defined forward the corresponding requests to the user's home institution. Apart from them, the proposed architecture (shown in Figure 3) also introduces a PEP in order to protect the resources, enabling the access to the authorized users, and a PDP to take the authorization decisions based on the defined XACML policies. The PDP also needs the user's attributes to evaluate these policies, which are harvested through the Attribute Requester for remote users and directly from the *User's database* for local users.

It is worth noting that the protected resource in the example shown in Figure 3 is the network, so that the PEP is the *Network Access Point* (NAP) that provides both wireless connectivity and access control capabilities according to the PDP's decisions.

Moreover, in this architecture we include two different repositories to store all the institution policies:

- *System Repository*. This contains both the traditional policies of the institution, e.g., access control policies or privacy policies, and the new administrative policies. All of them are created and managed by the system administrators.
- *Delegate Repository*. This repository contains all the access control policies generated by the delegates.

The reason of maintaining two different repositories is mainly the prevention of some security issues. The first repository, with the system policies, contains critical policies for the correct operation of the institution. This means that granting the access to other people may cause security risks.

Once the *Delegate* has been enabled as the person in charge of defining some system policies on behalf of the administrator, this user can access the *Template Server* to do this task. The Template Server is a Web application server that allows delegates to define access control policies from the privileges the system administrator has delegated on them. This component stores the templates, in the

form of Web-based forms, that delegates can fill out in a straightforward manner without having technical skills about policies management.

To do so, the Template Server maintains a third repository, apart from the two ones introduced above, where the templates are stored. These templates are automatically generated from the administrative policies created by the system administrators. Thus, delegates will then access to this server when they need to create an access control policy, according to the delegation model presented in Section III.

The Template Server is not very different to any other service that the federation offers, since this server will also make use of the defined mechanisms for controlling the access control to the templates. This decision is based on the credentials that the delegate owns in the institution to which belongs.

Therefore, delegates will have to authenticate in their home institution and, if both institutions belong to the same federation and delegates have the appropriate credentials, they will be able to access the Template Server.

The new components we have added in the identity federation system, shown at the top of Figure 3, can operate both in federated environments (inter-domain) and in autonomous institutions (intra-domain). The only difference between them lies on where and how the users' authentication and authorization are taken place.

In an autonomous mode of operation, all these processes are performed in the institution locally, whereas in a federated environment, as the one shown in Figure 3, the authentication is performed remotely at the user's home institution through *eduroam* and the authorization is carried out in the visited institution in a local way.

In the latter case, the authorization phase needs the user's attributes for the decision-making process. The visited institution should then interact with the user's home institution (note that both of them have to belong to the same federation) for recovering these attributes through *eduGAIN*.

D. Delegation policies

In order to take advantage of the administrative delegation, which is defined in the XACML guidelines detailed in [13], we have to define two different kinds of policies. On the one hand, an administrative policy expressing the delegation of the administrative rights to the delegates and, on the other hand, the access policies created by the delegates containing the details of the access control rules.

Using the use case presented in Section II, we have defined two example policies. The first one, the administrative policy, shown in Listing 1, specifies that a user who holds the *schacUserStatus* attribute with the *meeting:set* value can act as a delegate, and grant *Access* to the *Network* to any user who holds the *schacPersonalPosition* attribute with the *Researcher* value.

This policy also specifies that the maximum delegation depth is set to 1, by means of the *MaxDelegationDepth* attribute. Moreover, this administrative policy specifies some network properties to enforce; in this case, a QoS assurance with the *Class 2* value.

```
<Policy PolicyId="AdministrativePolicy" RuleCombiningAlgId="permit–overrides"
      MaxDelegationDepth="1">
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="string–equal">
          <AttributeValue DataType="string">Researcher</AttributeValue>
          <AttributeDesignator AttributeId="schacPersonalPosition"
                Category="...:delegated:...:subject–category:access–subject"/>
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match MatchId="string–equal">
          <AttributeValue DataType="string">Network</AttributeValue>
          <AttributeDesignator AttributeId="...:resource–id"
                Category="...:delegated:...:attribute–category:resource"/>
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match MatchId="string–equal">
          <AttributeValue DataType="string">Access</AttributeValue>
          <AttributeDesignator AttributeId="...:action–id"
                Category="...:delegated:...:attribute–category:action"/>
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match MatchId="string–equal">
          <AttributeValue DataType="string">meeting:set</AttributeValue>
          <AttributeDesignator AttributeId="schacUserStatus"
                Category="...:delegate"/>
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
  <Rule RuleId="AdministrativeRulePermit" Effect="Permit">
    <Obligations>
      <Obligation FulfillOn="Permit">
        <AttributeAssignment AttributeId="QoS">
          Class 2
        </AttributeAssignment>
      </Obligation>
    </Obligations>
  </Rule>
</Policy>
```

Listing 1. Administrative policy

On the other hand, the access control policy example is shown in Listing 2. This policy is issued by *Alice*, who grants *Access* to the *Network* to *Bob*. This policy could also include some conditions and network parameters that will have to be enforced by the PEP. In this case, Alice establishes that this access is only permitted from 9h to 13h.

```
<Policy PolicyId="AccessPolicy" RuleCombiningAlgId="permit–overrides">
  <PolicyIssuer>
    <Attribute AttributeId="...:subject–id">
      <AttributeValue DataType="string">Alice</AttributeValue>
    </Attribute>
  </PolicyIssuer>
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="string–equal">
```

```
            <AttributeValue DataType="string">Bob</AttributeValue>
            <AttributeDesignator AttributeId="...:subject–id"
                  Category="...:subject–category:access–subject"/>
          </Match>
        </AllOf>
      </AnyOf>
      <AnyOf>
        <AllOf>
          <Match MatchId="string–equal">
            <AttributeValue DataType="string">Network</AttributeValue>
            <AttributeDesignator AttributeId="...:resource–id"
                  Category="...:attribute–category:resource"/>
          </Match>
        </AllOf>
      </AnyOf>
      <AnyOf>
        <AllOf>
          <Match MatchId="string–equal">
            <AttributeValue DataType="string">Access</AttributeValue>
            <AttributeDesignator AttributeId="...:action–id"
                  Category="...:attribute–category:action"/>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
    <Rule RuleId="AccessRulePermit" Effect="Permit">
      <Target/>
      <Condition>
        <Apply FunctionId="function:and">
          <Apply FunctionId="function:time–greater–than–or–equal">
            <EnvironmentAttributeDesignator AttributeId="current–time"/>
            <AttributeValue>Wk0900</AttributeValue>
          </Apply>
          <Apply FunctionId="function:time–less–than–or–equal">
            <EnvironmentAttributeDesignator AttributeId="current–time"/>
            <AttributeValue>Wk1300</AttributeValue>
          </Apply>
        </Apply>
      </Condition>
    </Rule>
</Policy>
```

Listing 2. Access control policy

E. Conditions and obligations

As seen before, both policies define two sets of conditions and obligations (one per policy), which should be managed by the PDP. Thus, these conditions have to be combined each other and check them later to know they do not enter in conflict, or they are not contradictory. For example, the administrative policy could indicate that the user can only access the network from 8h to 14h, whereas the access control policy created by a delegate can restrict this period of time from 9h to 13h. In the policy evaluation process, which is carried out by the PDP, the conditions of the access control policies will be checked first, and the ones of the delegation policies later. As a result, the intersection of both constraints will be enforced, which is a correct way of operation.

On the other hand, the network parameters can produce some kind of inconsistency if the ones included in the access control policy are inconsistent with the ones established in the administrative policy. Although the correct way of managing the different kinds of network parameters depends on their type, in general, the values derived from the access policies can only be a subset of the ones derived from the administrative policy.

In this sense, delegates cannot grant wider privileges than the ones specified by the administrator in the administrative policy. In case of conflict between the properties stated by both policies, the PDP must ensure that the properties provided by the access control policy will be enforced iff they are less or equal than the ones permitted by the administrative policy.

V. Automatic Generation of Policies and Templates

The first step in the system is to define the administrative policy. This is done by the system administrator in a usual way, although in order to make this task easier the administrator could use any kind of XACML policy editor.

Among the available policy editors, all of them developed under an open source license, we can stand out:

- XACML-Studio [18]. This is an authorization policy editor implemented as a Web application to import, create, edit and export polices in XACML 2.0 format.
- XACML.NET [19]. This editor is completely developed in .NET/C#, which implements almost the entire XACML 1.0 standard, excepting the *regexp-string-match* function.
- eXist [20]. eXist-DB is a database capable of storing and managing information in XML format in a native way, as well as processing XQuery and XPath queries on the database itself. The policy editor is embedded within the editor itself that eXist provides to manage all the policies stored in its database. Anyway, this supposes a dependency drawback since it forces our infrastructure to use such a database to manage the policies from this editor. Moreover, it only supports the versions 1.1 and 1.0 of the XACML format.
- UMU-XACML-Editor [21]. This editor is implemented in Java with the aim of creating policies by following the XACML 2.0 standard.

Although UMU-XACML-Editor, as previous ones, does not support the latest XACML specification (version 3.0) we have chosen it as the policy editor for this research work.

This solution is the policy editor most updated of the existing ones, in addition to some interesting technical features such as defining references to other policies and supporting the *SchemaLocation* element to validate polices against their XML schemes. Thus, the UMU-XACML-Editor policy editor has been extended to support the new XACML delegation profile, as presented in Section III.

Once the administrator has defined the administrative policy by using the UMU-XACML-Editor, the next step is the definition of the access control policies by the delegate. But as indicated previously, how the delegate is going to create the access control policies is a tricky aspect, because she is a common user and not a security expert with skills on policy languages.

Therefore, instead of building this policy from scratch, some kind of template (in our case, Web-based forms) can be provided to the delegate to help her in this administrative

task. In this way, the delegate will only have to fill in this template to create the appropriate access control policy.

A. Generation of templates

To perform this process is necessary to use a *Policy Management Tool* (PMT) with the aim of building the access policy templates. In our case, we have decided to extend the UMU-XACML-Editor policy editor to include this new feature. This tool will use an XSLT [22] transformation that extracts the appropriate fields from the delegation policy to generate the template.

The PMT needs to search for the *action* and *resource* elements specified in the administrative policy, which are identified by the "*...:delegated:...:attribute-category:action*" and the "*...:delegated:...:attribute-category:resource*" categories (see Listing 1). The rest of data needed for the access control policy, such as the subject identifier, are included in the template as input fields for being filled in by the delegate. The identifier of this delegate is automatically added in the new policy as the issuer thereof.

This process is depicted in Figure 4. Note that the text enclosed in parentheses indicates which element of the infrastructure (the administrator, the delegate or the system itself) generates each piece of information.

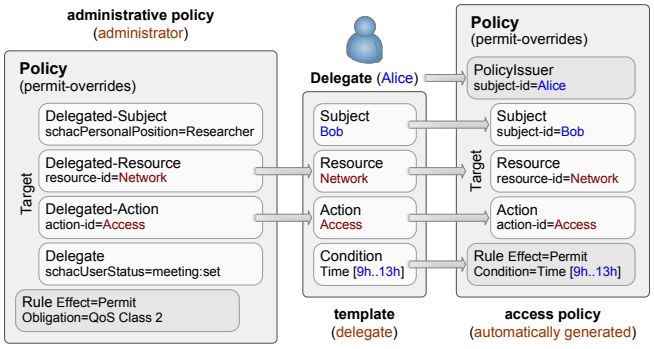

Figure 4. Administrative delegation process

Besides, the template must allow the delegate to specify the conditions and obligations. The most common condition is the time constraints although, however, other conditions such as a maximum number of connected users could be included in it. Regarding the network parameters, the template can also show to the delegate the different parameters defined in the system that she can assign, e.g., QoS or bandwidth, together with their possible values.

Once the PMT has generated this template, it is necessary to make it available to the delegates on some server or repository. But the access to this template must be restricted to the appropriate delegates, so that the PMT should also generate an XACML policy to control the access to it. In this case, the PMT must search for the "*...:delegate*" category in the administrative policy, which specifies the restrictions for the delegates. This field is then used to specify the subject of the policy to control the access to the template. Finally, both the administrative policy and the access control policy to the template are stored in the System Repository to be used later.

As we can see in Figure 5, from the administrative policy created by the administrator, the PMT is also capable of: (1) generating the template for the delegate; and (2) generating the corresponding policy to control the access to such a template. Both policies are stored in the System Repository with the system policies, whereas the template is stored in the internal repository of the Template Server.

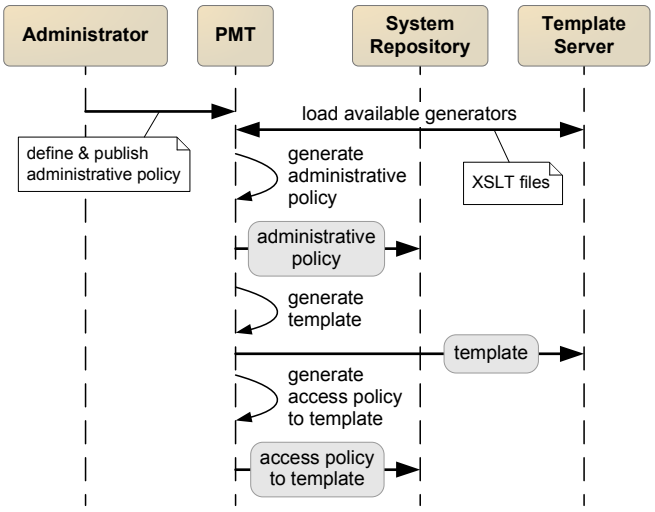

Figure 5. Generation of automatic templates

In order to control the access to these templates, the Template Server has to query the PDP to take an authorization decision according to the *access control policies to templates* previously generated by the PMT.

B. Generation of access control policies

Figure 6 shows the complete process for the generation of access policies from the previously generated templates. Before allowing delegates to access to these templates, the Template Server should determine whether the delegate owns the required credentials and privileges to take advantage of an administrative delegation.

To do so, the delegate must first be authenticated through the AuthN module (see Figure 3). At this point, it is worth pointing out that this user will be authenticated in her home institution, either locally if the delegate belongs to the same institution or remotely through *eduroam* if she belongs to a remote institution with which the visited institution has a close relationship through the same identity federation.

If the authentication has been successful, the Template Server has to collect the delegate's attributes to decide if this user owns the minimum privileges to complete this administrative task. This harvest process is performed through the Attribute Requester module (see Figure 3), which will

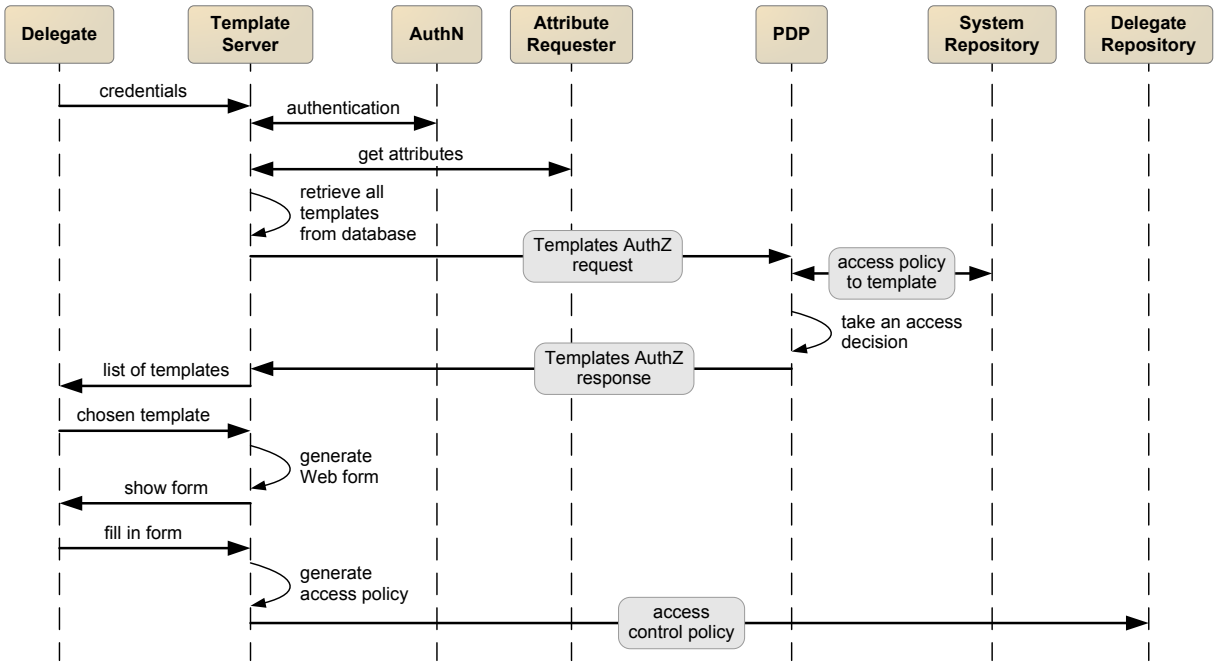

Figure 6. Process of using templates by the delegates

gather them from the home institution to which the delegate belong. As before, this process will be done either locally or remotely (in this case, through *eduGAIN*), depending on which the delegate's institution is.

After this harvesting process, the Template Server retrieves all the defined templates from its internal repository. Then, the Template Server builds an authorization decision request that sends to the PDP to know which of these templates the delegate can fill in.

```
<Request>
  <AuthorizationDecisionQuery Resource="Template-11">
    <Subject>
      <NameIdentifier NameQualifier="https://idp.um.es/shibboleth/idp">
        Alice
      </NameIdentifier>
    </Subject>
    <Action Namespace="urn:segura:umu:actions">Fill-in</Action>
    <Evidence>
      <Assertion>
        <AttributeStatement>
          <Attribute AttributeName="eduPersonPrincipalName"
                AttributeNamespace="urn:segura:edugain:attributes">
            <AttributeValue>alice@um.es</AttributeValue>
          </Attribute>
          <Attribute AttributeName="schacUserStatus"
                AttributeNamespace="urn:segura:edugain:attributes">
            <AttributeValue>meeting:set</AttributeValue>
          </Attribute>
        </AttributeStatement>
      </Assertion>
    </Evidence>
  </AuthorizationDecisionQuery>
</Request>
```

Listing 3. Access request to verify if the delegate can fill in a template

This authorization decision request must include:

- the attributes of the delegate in the *Assertion* element;
- the template to be accessed in the *Resource* element;
- and the *Action* element with the *Fill-in* value.

An example for the use case presented in Section II can be found in Listing 3. In this case, the PDP is evaluating whether Alice can fill in the template number 11 or not. If so, the PDP will return to the Template Server a signed authorization decision response stating that *Decision="Permit"* to the *Resource="Template-11"* for the *Action="Fill-in"*.

During this decision-making process, the PDP makes use of the access control policies to templates, which have been previously stored in the System Repository by the PMT.

Finally, the Template Server will show to the delegate the list of templates that can fill in according to the presented credentials. A screenshot of this Web page can be found in Figure 7. At the top of this figure is shown the graphical result obtained by Alice after entering her credentials in a previous phase. In this case, Alice can only fill in the template number 11 with the "*Meeting Segur@ Internet Connection*" description. At the bottom of the same figure, we can see the logging server output to check the different steps explained in this section.

Once the delegate selects one of the available templates, the Template Server internally generates the Web form that will return her so it can be finally filled in. Continuing the previous example, Figure 8 depicts another screenshot where Alice has already filled in the Web form with the requested information. As can be seen, the administrative definition of a new access control policy is a very straightforward process that does not require special skills in managing policies.

Finally, and after filling in the template, the Template Server internally generates the access control policy and

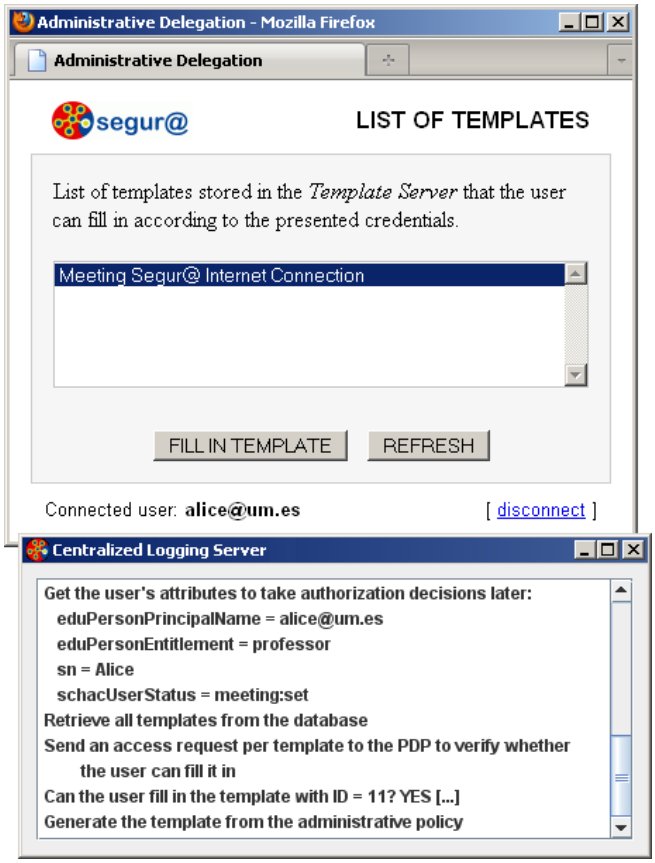

Figure 7. List of templates the delegate can fill in

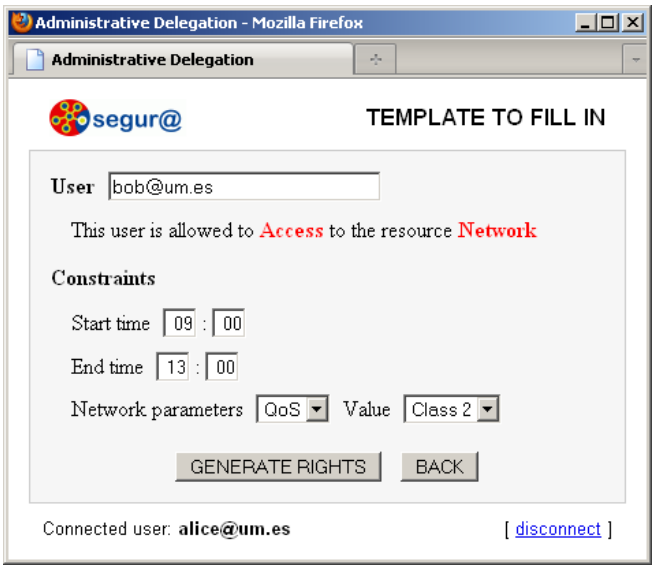

Figure 8. Template filled in by the delegate

stores it in the Delegate Repository for uses in future requests from *Bob* to *Access* to the *Network*, provided this request is from 9h to 13h. For this generation we also make use of XSLT transformations, as seen in the right-hand side of Figure 4.

VI. Performance Results

The infrastructure with delegation support proposed in Section IV has been implemented and deployed in a lab testbed to demonstrate its applicability in a real scenario.

In these tests, we have assumed that both the administrator and the delegate belong to the same institution, so no authentication process is carried out remotely through *eduroam*. As a consequence, the remote harvesting process for getting the delegate's attribute is not performed through *eduGAIN*. In its stead, all these processes are taken in the same institution.

This validation has been performed through some performance measurements, which have been taken directly from this testbed, depending on different factors that might have an important impact on the proposed administrative delegation process. These tests are:

- Retrieve the list of templates from the Template Server that a delegate can fill in. In this process, we also include the authentication and authorization phases of the delegate when she accesses to the Template Server for first time.
- Validation of an access control policy sent to the PDP by a user.

In both cases, we have assumed for these tests our infrastructure is configured with the policies and features presented throughout this paper. The list of hardware and software requirements deployed in our lab testbed is shown in Table I.

A. Retrieve the list of templates from the Template Server

This first test aims to assess the time a delegate needs to access the Template Server in order to fill in some of the templates for which she is responsible. It will give us an idea about the time consuming on the different phases involved in this process, as detailed in Section V-B, which have been divided in four different steps (all of them corresponding to the arrows depicted in Figure 6):

- *User AuthN*. Time needed by the infrastructure to authenticate the delegate. This step corresponds to the *authentication* row depicted in Figure 6.
- *User AuthZ*. This step represents the time needed to retrieve the delegate's attributes by means of the Attribute Requester. This corresponds to the *get attributes* arrow.
- *DB Templates*. It corresponds to the *retrieve all templates from database* arrow. This retrieval is internally done from the internal repository of the Template Server.
- *Templates AuthZ*. This step pertains to the decision-making process that the PDP has to do to know which templates the delegate can fill in.

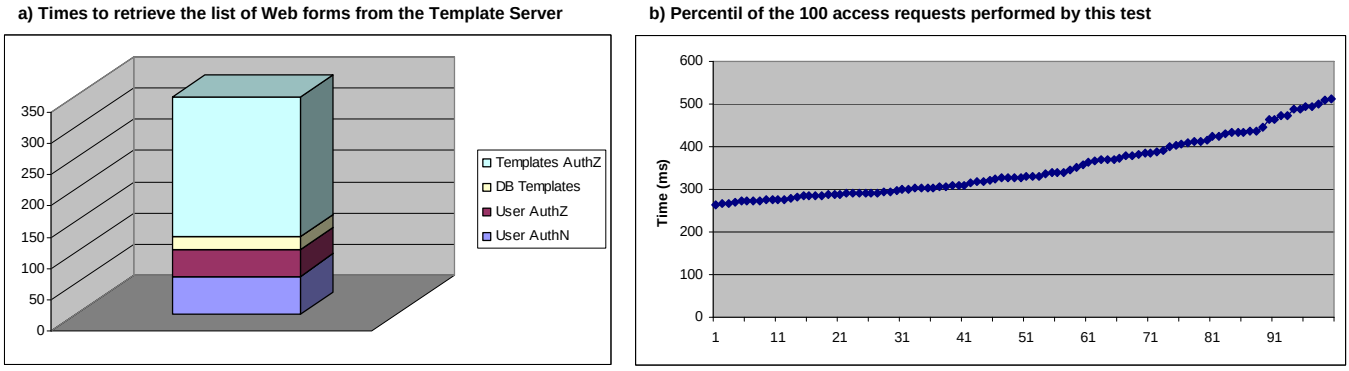

Figure 9. Administrative delegation process

	Hardware	Software
Template Server	Intel Pentium 4 CPU 640 CPU: 3.20 GHz, 32 bits Cache size: 2048 KB L2 Total memory: 1024 MB	Windows XP SP3 Tomcat 6.0.10
PDP	AMD Opteron CPU 246 CPU: 2.0 GHz, 64 bits Cache size: 1024 KB L2 Total memory: 1024 MB	Ubuntu 9.10 Tomcat 5.5.27 eXist-DB 1.2.6
AuthN & IdP		Ubuntu 9.10 Apache 2.2/mod_ssl MySQL 5.0.75 OpenLDAP 2.3.28
Attribute Requester		Ubuntu 9.10 Tomcat 5.5.27

Table I
HARDWARE AND SOFTWARE REQUIREMENTS

The testing process has been performed by means of sending 100 sequential requests, from which we have extracted the average times of each these four steps. Note that all times have been measured in milliseconds (ms). The partial times of each of them are shown in Figure 9a, while Figure 9b illustrates the total times for the 100 requests.

We can observe in Figure 9a what is the time distribution for each case. At first sight, we can assert that the decision-making process performed by the PDP is the longer time of all, as expected, being the 63,81% of the total time. The *Templates AuthZ* step takes 224 ms, on average, for the 100 tests performed, while the total time takes 351 ms. For this process, the PDP only needs to evaluate the access control policies to templates, so no delegation chain is used by it.

It is worth noting that these times have been taken from the Template Server. This means that the PDP does not really take these 224 ms, because we have to take in consideration the network traffic and delays. We have measured this time, and the PDP really takes 198 ms on average in this process of making a decision. With this last time, it would be now a 54% of the total time. Apart from that, this time is very reasonable since the PDP has to take its decision, build the corresponding SAML response and sign it digitally.

In any case, as a conclusion, these times are perfectly acceptable and assumed by the delegate.

B. Validation of access control policies

As have seen in the previous test, the PDP is the critical component that can slow down the performance of the overall system. Then, for this second test we have taken into consideration how the delegation chain defined in Section III can influence in this performance.

That delegation chain stated that the administrator delegated to Alice the definition of the access control policies for such a scenario. Thus, once Bob tries to access the network the PDP should check both the administrative policy, created by the administrator, and the access control policy created by Alice. In this process, the PDP will also have to retrieve the Alice's attributes for making the corresponding decision. Note that the retrieval of these attributes will be done through the Attribute Requester.

For this test, as before, we have performed 100 sequential request. The results for this test can be found in Figure 10.

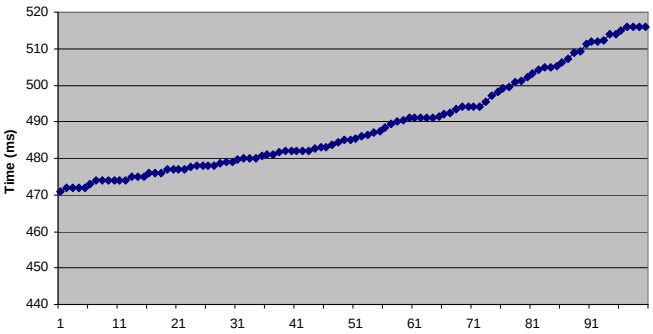

Figure 10. Percentile times for validating a delegation chain

In this case, the PDP takes 489 ms on average to validate the complete delegation chain. This process includes: the retrieval of both policies from their corresponding repositories, the administrative policy from the System Repository and the access control policy from the Delegate Repository;

the harvesting process of the Alice's attributes through the Attribute Requester component; the internal validation from both policies; and, finally, the building and digital signing of the SAML response.

Among these steps, the largest computational load corresponds to the retrieval of the policies from their repositories, taking 196 ms on average. This supposes the 40% of the total time. Thus, and depending on the scenario we are considering, some optimizations could be implemented, e.g., by caching these policies in the PDP, to reduce these times in a real production system.

VII. Related work

The XACML delegation profile specified in [13] is very recent, so there are not many works making use of it. However, the idea of delegation of rights has existed for several years, and there are a lot of works trying to include this feature in different ways. For example, in PKI systems, two initiatives stood out in the use of delegation [23]: SPKI/SDSI defines a standard way for digital certificates whose main purpose is authorization rather than authentication; and X.509 proxy certificates are another way of providing restricted proxying and delegation within a PKI-based authentication system.

In the latest years, authorization systems are making use of XACML because it is standard, and besides it provides an expressive access control language. Before the apparition of the XACML delegation profile, several proposals to add these features to XACML were made. For example, in [24] the authors presented a system permitting controlled policy administration and delegation using XACML in combination with a second access control system. This system is *Delegent*, which has powerful delegation capabilities. But contrary to our proposal, this system does not define how delegates manage the XACML policies.

Another proposal to add dynamic delegation to XACML is presented in [25]. But unlike the XACML delegation profile, this system is based on the delegation of roles instead of the delegation of policy administration. This system defines a validation service whose purpose is to validate a set of credentials for a subject, issued by multiple dynamic attribute authorities from different domains. Finally, it returns a set of valid attributes that are used in the XACML system in the standard way.

VIII. Conclusion and future work

This work shows that although the administrative delegation is a helpful tool for managing policies in complex systems, it also introduces some drawbacks that hinder its use in real existing environments. Therefore, this paper describes an infrastructure that makes use of the administrative delegation in an effective way, thus simplifying the work of both the system administrators and the delegates. On the one hand, the workload of the administrators is reduced by distributing the policy management among the appropriate users in the system, i.e., the delegates. On the other hand, the delegates, who are not concerned about policy management, only have to fill in the appropriate templates to generate these policies in an automatic way.

Currently, as a statement of direction, we pretend to study in depth the management of delegation chains in highly distributed authorization systems.

Acknowledgment

This work has been partially funded by the CENIT Segur@ (Seguridad y Confianza en la Sociedad de la Información) project and the MASTER EU-IST project (IST-2001-34600) within the EC Seventh Framework Programme (FP7). Authors would finally like to thank the Funding Program for Research Groups of Excellence with code 04552/GERM/06 granted by the Fundación Séneca.

References

[1] M. Gil Pérez, G. López, A.F. Gómez Skarmeta, and A. Pasic. "Advanced Policies for the Administrative Delegation in Federated Environments". In *DEPEND'10: Proceedings of the 3rd International Conference on Dependability*, pages 76–82, July 2010.

[2] The ERASMUS Programme Web site, European Commission. http://ec.europa.eu/education/programmes/llp/erasmus 25.05.2011.

[3] Confederation of EU Rectors' Conferences and the Association of European Universities (CRE). "The Bologna Declaration on the European Space for Higher Education: an Explanation", June 1999.

[4] European Higher Education Area (EHEA) Web site 2010-2020. http://www.ehea.info 25.05.2011.

[5] K. Wierenga and S. Winter (main editors). "Inter-NREN Roaming Architecture: Description and Development Items". GÉANT2 JRA5, Deliverable DJ5.1.4, September 2006.

[6] Haka Federation Web site, CSC-IT Center for Science Ltd. http://www.csc.fi/english/institutions/haka 25.05.2011.

[7] SWITCH Federation Web site. http://www.switch.ch/aai 25.05.2011.

[8] M. Sánchez, O. Cánovas, G. López, and A.F. Gómez Skarmeta. "Levels of Assurance and Reauthentication in Federated Environments". In *EuroPKI'08: Proceedings of the 5th European PKI Workshop on Public Key Infrastructure: Theory and Practice*, pages 89–103, June 2008.

[9] E. Rissanen and B.S. Firozabadi. "Administrative Delegation in XACML - Position Paper". Swedish Institute of Computer Science, September 2004.

[10] A. Pasic, J. Bareño, B. Gallego-Nicasio, R. Torres, and D. Fernandez. "Trust and Compliance Management Models in Emerging Outsourcing Environments". In *Software Services for e-World*, volume 341 of *IFIP Advances in Information and Communication Technology*, pages 237–248. November 2010.

[11] The MASTER EU-IST Project (Managing Assurance, Security and Trust for Services). http://www.master-fp7.eu 25.05.2011.

[12] T. Moses (editor). "eXtensible Access Control Markup Language (XACML) Version 2.0". OASIS Standard, February 2005.

[13] E. Rissanen (editor). "XACML v3.0 Administration and Delegation Profile Version 1.0". Committee Specification 01, August 2010.

[14] J. Masa (editor). "SCHema for ACademia (SCHAC): Attribute Definitions for Individual Data Version 1.4.0". Working Draft, March 2009.

[15] Education Roaming (eduroam) Web site, TERENA Association. http://www.eduroam.org 25.05.2011.

[16] D.R. Lopez (main editor). "GÉANT2 Authorisation and Authentication Infrastructure (AAI) Architecture Second Edition". GÉANT 2 JRA5, Deliverable DJ5.2.2,2, April 2007.

[17] Education GÉANT Authorisation Infrastructure (eduGAIN) Web site. http://www.edugain.org 25.05.2011.

[18] O. Gryb. "XACML-Studio (XS) Reference", November 2009. http://xacml-studio.sourceforge.net 25.05.2011.

[19] D. González and A. Neisen. "XACML.NET Version 0.7", March 2005. http://mvpos.sourceforge.net 25.05.2011.

[20] M. Harrah. "Access Control in eXist", September 2009. http://exist-db.org/xacml.html 25.05.2011.

[21] University of Murcia. "UMU-XACML-Editor Version 1.3.2". http://sf.net/projects/umu-xacmleditor 25.05.2011.

[22] M. Kay (editor). "XSL Transformations (XSLT) Version 2.0". W3C Recommendation, January 2007.

[23] M.R. Thompson, A. Essiari, and S. Mudumbai. "Certificate-Based Authorization Policy in a PKI Environment". *ACM Transactions on Information and System Security (TISSEC)*, 6(4):566–588, November 2003.

[24] L. Seitz, E. Rissanen, T. Sandholm, B.S. Firozabadi, and O. Mulmo. "Policy Administration Control and Delegation Using XACML and Delegent". In *GRID'05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 49–54, November 2005.

[25] D.W. Chadwick, S. Otenko, and T.-A. Nguyen. "Adding Support to XACML for Dynamic Delegation of Authority in Multiple Domains". In *CMS'06: Proceedings of the 10th IFIP TC-6 TC-11 International Conference on Communications and Multimedia Security*, pages 67–86, October 2006.