Deductive Policies with XACML

Mario Lischka, Yukiko Endo  
NEC Laboratories Europe  
NEC Europe Ltd.  
Heidelberg, Germany  
(lischka|endo)@nw.neclab.eu

Manuel Sánchez Cuenca  
Department of Information  
and Communications Engineering  
University of Murcia  
Murcia, Spain

ABSTRACT

SaaS technology might comprise of a bundle of different services provided by different entities. Thus monolithic access policies are not feasible as each of the service partners and the companies using the service would have to provide their internal and potentially confidential rules on which they base their policies. In addition internal information such as concrete position of the user or affiliation to a specific project might be utilized in the policies and should not be provided to any external entity.

Deduction of decisions has been investigated for more than a decade, but no widely spread standard has been defined, so far. OASIS XACML is being used in many applications and services nowadays. Additionally, tools for modeling the policies are available and many engineers share common understanding of this approach. In this paper we present an extension of the XACML language to support deduction of decisions, together with a distributed definition of the policies and at the same time avoiding problems known from current solutions on deductive policy languages.

1. INTRODUCTION

Instead of using specific application on a local system, companies are starting to outsource part of their IT infrastructure using SaaS technology which might actually consist of a bundle of different services. As these services might be provided not only by the company itself but also by different service providers, a monolithic access policy is not feasible. Each of the service partners and the companies using the service would have to provide their internal and potentially confidential rules on which they base their policies. In addition, the companies internal information such as concrete position of the user or affiliation to a specific project might be utilized in the policies and should not be provided to any external entity.

To this effect, we are introducing deductive policies which allow us to specify policies at the SaaS service to deduce the decision whether a particular user might use the service based on (access) requests forwarded to the underlying services and/or the particular company associated with the user. Based on this approach it is not required to define a separate and independent set of policies for each domain (e.g. service), which might contain the risk of inconsistencies between them. The related deduction policies might be specified in a general manner describing the translation from one domain to the other. These policies do not handle the details of the actual access conditions, as the deduction provides an abstraction layer. Therefore, the length of such policy tends to be shorter, and their structure less complex. Thus it is much easier for an administrator to analyze the policy and verify its correctness.

This approach requires a policy language and a policy framework which supports deduction. OASIS XACML is used in many applications and services nowadays [10, 11], its concepts are, also more targeted for a central, monolithic definition of the policies. Having in mind a distributed environment and a cross layer approach – as it is currently developed in the SWIFT project [9,12] – the current XACML, including the latest draft [11], approach is not feasible, as it only allows to combine the results with others on the very same request. Thus, no changes in the subject or the resources could be modelled in the policy in order to refer to other types of applications. In addition the combination algorithms provided in the XACML specification are quite limited, targeting mainly overriding approaches or ordered preferences, but no support for binary expressions is given.

As a better solution, we propose that an entity must be able to deduce its decision from arbitrary results of other entities. These entities are domains which have specified a set

Categories and Subject Descriptors
D.4.6 [Security and Protection]: Access controls; D.4.7 [Organization and Design]: Distributed systems

General Terms
Security

Keywords
XACML, Deduction

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

©2009 ACM ...$10.00.
of policies. The related evaluations are done at a dedicated policy decision point, which could be queried by other PDPs to deduce decisions. As mentioned before, an important aspect is that the subject and the object of the derived requests may be changed according to the specification of a policy.

In order to achieve this, we have developed a policy language which supports deduction of decisions, together with a distributed definition of the policies. This policy language is defined as an extension of the current draft of XACML 3.0 [11], but it might also be applied to XACML 2.0 [10] with minor changes.

In the following subsection we will give a more detailed example scenario illustrating the need for this kind of policies. In Section 2 we will present the related work in this area. The concept of our approach is discussed in Section 3 and its realization is introduced in Section 4. In Section 5 we will discuss the complexity of the proposed extension towards deductive policies. We will show in section 6 how the scenario presented below could be realized through policy based on our proposed extension to XACML and Section 7 concludes this paper.

1.1 Application Scenario

As an illustrative example let’s assume that an employee of a company wants to use a SaaS platform for multimedia data. While the core functionality is provided by the provider of the SaaS platform additional services such as watermarking – whose hidden information allows the company to prove its origin later – are provided by others. The employee’s company wants to limit the access to this SaaS service not only to a particular set of its work force but also keep control of the material (i.e. the original sources) used for this service, e.g. avoid highly confidential material from being edited on such a SaaS platform. The respective policies should be modeled by the company itself, as internal project structures as well as information on the resources are considered confidential. In addition, depending on the material specific watermarking technologies have to be applied to the multimedia files. The SaaS integrates the different policies through generic deductive policies, ensuring that together with its own policies, the policies of the company are respected and privileges to use the external services (e.g. for watermarking) are actually available. For the latter it should be sufficient that at least one of the watermarking services is available with the capability required by the company. This required capability will be explicitly requested from the company.

This procedure is sketched in Figure 1. One might observe that neither the internal policies of the company nor those of the additional services have to be known at the SaaS service. Thus changes as well as the underlying rules or attributes are kept internal. The details of the actual policy set of the SaaS service to specify this example are discussed in Section 6.

2. RELATED WORK

Different approaches than deduction have been proposed to support access control in a distributed environment. In [3] a delegation mechanism is presented for a multi domain environment, but problems may arise if policies are relying on up-to-date information, which should be kept confidential within a domain. In case of policies that are stuck to the data and could be evaluated at different locations as proposed in [4] the same problem with respect to attributes referred in the policy arises. In [5] the coordination of distributed policies decision is not explicitly done through the policy itself, instead each local policy is checking on a global, coordinating variable. As this paper presents an extension to XACML for deductive policies we will first describe the related work in the area of deduction of decisions, before we give a short overview on the current specification of XACML.

2.1 Deductive Policies

Datalog has been introduced in the area of databases as a method to gain more general knowledge from a base of different facts. As a predicate logic it provides methods to deduct new facts from existing ones. An important aspect is the definition of a predicate rule which contains a negation of a predicate. This rule is evaluated as true, if negated predicate is evaluated as false. Due to the dependencies among the predicates this type of negation, some problems related to decidability arise, in case circular dependencies occur. This problem has been examined and solution based on stratification of the predicates has been proposed (among others in [14]).

Datalog as a method to express access control policies has been discussed over the last decade (see [7] for the basic concept). In [15,16] the concept was extended to support policies specified independently by different organizational entities and resolve potential conflicts among them. In [1] an algebra was presented to solve potential conflicts for non hybrid access policies, and a method to stratify predicates defined by different domains has been presented. Hybrid policies and their algebraic combination have been presented in [2] based on a four value logic. In [8] organizations are reflected in the policies. But it is unclear how the dependencies between policies defined by the different organizations are referred to or resolved. Both points are addressed in [15,16] and will be covered in this paper as well.

2.2 OASIS XACML Policy Language

OASIS XACML (eXtensible Access Control Markup Language) [10,11] was defined to represent access control policies in a standard way. XACML is XML-based and includes two different specifications: the first one is an access control policy language, which defines the set of subjects that can perform particular actions on a subset of resources; the second one is a representation format to encode access control requests and responses, that is, a way to express queries

---

![Figure 1: Deductive decision on access to an SaaS service](image-url)
about whether a particular access should be allowed and the related answers.

The main element of all XACML policies is a Policy or PolicySet element. A PolicySet is a container that can hold other Policies or Policy Sets, as well as references to other policies (Policy). A Policy represents a single access control policy, expressed by a set of Rules. A Policy or PolicySet may contain multiple policies or Rules, each of which may evaluate to different access control decisions. In order to recognize these decisions, XACML has a collection of combining algorithms. An example of those is the Permit-Overrides algorithm, which says that if at least an evaluation returns Permit, then the final result is also Permit. A Policy or PolicySet element may also specify a set of Obligation Attributes, which is a set of actions to be performed depending on the decision.

With the purpose of determining if a specific PolicySet, Policy or Rule is applicable for a given request, XACML defines the Target element. It is a set of simplified conditions for the Subject, Resource and Action, which must be met in order to apply its associated PolicySet, Policy, or Rule to the request. Once a Policy is found, the included rules are evaluated. The main element of a rule is the Condition. If the evaluation of the Condition results true, then the Rule’s Effect (Permit or Deny) is returned.

3. GENERAL CONCEPT OF DEDUCTIVE POLICIES

Some access policies have the tendency to grow more complex over time, as they have to be adapted to deal with additional cases and still cover the original one. Other policies e.g. related to (generic) privacy policies have to be fulfilled by organizations due to legal regulations are quite complex right from the beginning. In enterprises, different divisions might have their own (complex) policies instantiated.

In addition to this, two additional reasons come into play. On the one hand these services could be composed out of various elements, thus access to the services as a whole might be bound by access privileges granted by all the providers of these elements. On the other hand, although the SaaS service might be outside of the original domain of an organization, parts of the access privileges might have to be kept under its control perhaps due to the reasons stated at the beginning of this section. Therefore, a monolithic policy would be complicated. The key point of Deductive Policies is to introduce a kind of abstraction or modularization, breaking the policies into different building blocks. We call this the Authoritative Domain which consists of several policy sets defined by a particular entity. Inside these specific aspects e.g. generic data privacy could be specified. Due to the proposed extensions to XACML other policies could refer to this set of policies without having to know the details of the policies given inside. As each of them might have their own notion of subject, resources and action an important aspect is the mapping of the latter between different sets of policies. Therefore, we must be able to transform the given request in a particular environment of domain A into one with a (slightly) different context of another authoritative domain B. Depending on the results of the domain B the result of domain A is deduced as sketched in Figure 2(a). It is important to notice that the domain might not be fixed, but instead be dynamically determined e.g. based on attributes of the original request. An example is indicated in Figure 2(a), where policy set B2 is either deducting its decision on the results of the authoritative domain D or E, depending on the resource of the request to B. In the same way several domains may give a result on some request, but do not provide a result on others.

Negation and non existing results is a well known problem of potential undecidability of rules, and it has to be tackled. Stratification of the domains or layers is one of the possible solutions which have been explored for deductive languages in general [14]. As the support of hybrid policies is essential, the problem of decidability with respect to negations inside a policy has to be well observed. Thus circular dependencies among the authoritative domains as shown in Figure 2(b) have to be avoided. If this is not the case, the results on the circular path switches when evaluated again depending on a negation or the non-existence of a result. Thus the final results becomes undecidable.

Based on the research in the area of Datalog we decide to take a stratified approach, but instead of the predicates the authoritative domains have to be ordered. One of the requirements of our language will be that a deduction is only taken from a subordinated domain. These subordinated domains have to be explicitly defined (details will be presented in 4.1). Based on this a Distributed Depth-First Search could be used to detect circular dependencies. Thus, this is an important aspect, as the relation among these authoritative domains is well observed in our approach..

4. EXTENDING XACML TOWARD DEDUCTIVE POLICIES

Although the current version of XACML does not support a distributed decision making it is widely used to specify the access policies and supporting tools are available for modeling as well as analyzing these policies. After careful examination of the general underlying principles and the needs defined by the SWIFT project [9, 12] the current specification of XACML could be extended to support deduction among different layers and in a distributed environment. We will refer to this language as XADML or extensible authorization deduction markup language.

The key aspect of deductive policy is to include results from other (remote) policies into a local set of policies. As these policies will most likely refer to different domains, a method has to be found to define the related subjects and
resources, as well as specific actions independent of those given in the current policy. Here, additional or extended elements (with respect to the current XACML standard) are needed. In order to support specific reasoning regarding the existence of a policy, negation of the deduced policy has to be included in the specification of the policy language through new combining algorithms.

Although XACML has a notion to refer to policies, the responsible entity could not be identified in a unique fashion, as the element of PolicyIssuer may contain a unlimited number of content entries. The details on the relationship will be discussed in Section 4.1. Another important aspect is the exchange of additional information related to a policy decision as identified in the scenario in Section 1.1. This will supply the deducting entity with data that could be used by the conditions. Further details and the related extensions are discussed in Section 4.3.

The extensions towards a support of deductive decisions modeled in XACML should be as minimal as possible in order to ensure an easy adaption of existing interpreters as well as other tools. These new elements are indicated in Figure 3 with a grey background, while the existing XACML elements have a white one.

We have to redefine the existing element PolicySet in order to integrate our extensions. In the following sections we will see that the additional specifications for the policy language are straight forward. In the following section we will present the key features of the proposed extension in details. The full schema which defines our deductive policy language is presented in the Listings 1 to 4. Elements or types defined in XACML have the prefix xacml; while newly defined elements and types have the prefix xadml: in all of the following listings. The impact on the XACML framework will be discussed in Section 4.4.

4.1 References to other Policies

One of the key aspects of deductive policies is the reference to policies defined by another entity. In principle a set of policies belonging to another well distinct application area is considered for a different domain, although it might belong to the same organizational entity. As discussed in Section 2.1 circular dependencies among the predicates might lead to an undecidable request, illustrated in Figure 2(b). At best due to the circular dependency no result is given. At worst if the first result is found and the evaluation passes a second time the previous evaluation might get inverted due to a negation inside a policy rule. This would lead to an update of all the other results on the circle and then again a change due to a negated deduction.

One of the known solutions already mentioned is stratification [14], which avoids any circles by introducing a hierarchical dependency, originally among predicates, and in our case among policy rules. In an organizational context, which could be assumed for most authorization systems, this hierarchy can be achieved, by breaking large domains into smaller entities which then have no circular dependency.

Due to the potentially distributed definition of the policies, the dependencies among them have to be well observed, in order to ensure that a partial order exists. Therefore each deductive policy specifies the authoritative domains it is referring to. During loading of (new) policies a depth-first search algorithm with related queries could ensure that no circular dependency exists, among this policy and any of the referred ones. In a straightforward manner, this could be achieved with a Distributed Depth First Search by sending around a token to the PDP of the referred Authoritative Domain, which again forwards this token to its referred domain. Note, that this has to be done when the referred domains are changed, not during each evaluation.

Thus, for our deductive policies we have to require that the referred domains used for deduction are specified for each policy. In Section 4.2 we will allow variables, whose values are assigned during the evaluation of a policy, to specify the domain used for deduction. As analyzing the complete value space of these variables might be too complicated, these domains shall be stated explicitly during the definition of the policy set.

The related extension to XACML is shown in Listing 1. For each policy set a list of referred domains might be specified (see line 11). Alternatively, if this list is identical for a number of policy sets, they all may refer to file (see line 13), containing a shared list of referred domains. This file contains an element of DomainReferenceType (see lines 18 to 23), which contains the list of domains and a unique identifier for this list.

As stated before, the existing XACML 3 definition does

---

**Listing 1: References to Domains**

```
1 <xs:schema xmlns:xadml="urn:eu:xadml:schema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:ld-1"
  targetNamespace="urn:oasis:names:tc:xacml:3.0:core:schema">
  <xs:element name="DomainReferenceDefinitionType" type="xsd:extension"/>
  <xs:element name="DomainReferenceDefinition" type="xsd:complexType" maxOccurs="unbounded">
    <xs:sequence>
      <xs:element name="DomainReference" type="xsd:DomainNameType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="DomainReferenceList" type="xsd:complexType">
    <xs:sequence/>
  </xs:complexType>
  <xs:complexType name="DomainReferenceDefinitionType">
    <xs:attribute name="DomainReference" type="xsd:DomainNameType" maxOccurs="unbounded"/>
  </xs:complexType>
  <xs:complexType name="DomainReferenceType">
    <xs:attribute name="DomainReference" type="xsd:DomainNameType" maxOccurs="unbounded"/>
  </xs:complexType>
</xs:schema>
```
not have to be modified to a larger extent. A redefinition of PolicySet is required and shown in Listing 2, as the extension mechanism of XML does not ensure the desired results.

The domain which is responsible for this set is given in line 30. In case reference to other policies are included their specific domains have to be given either directly or indirectly in the policy (see line 13) as discussed above.

As mentioned in the introduction the PolicyIssuer (line 28) might not be unique as it could contain a set of identifiers. The attribute PolicyID (line 47) identifies a single policy, but we want to refer to a whole set of PolicySet elements specified by one domain.

4.2 Deducting Decisions

The key point for extending XACML to support deducting a policy decision based on the results of another policy. This kind of reference is done at the same level as a policy, due to two major reasons. First, the combining algorithms could be used to integrate the respective results including their obligations and advices in a well defined manner. As mentioned above some additional combining algorithms might be needed to handle indeterminate or non applicable references to other policies, like combine policies which provide a non-applicable result of the deductive policies into a deny result. Second, PolicySets could be used to structure these references into more complicated expressions.

The domain which is used for deducing the decision is either explicitly given by a fixed value or through a variable defined in the PolicySet (see line 31) and assigned during the evaluation. The latter allows us to refer to a domain, depending on the given request and other context information. Due to this required feature the definition of a variable has to be introduced inside of a policy set (see line 34). We assume that this variable is known in all Policy Sets and policies directly included.

During the evaluation of a concrete request it has to be verified by the PDP that only a reference to those domains is actually given through a variable, which have been defined

---

2The detailed definition of this element is given in Listing 1 and has been discussed in Section 4.1.

3We assume that in most cases only a subset of the attributes of the requests is reused.
4.3 Deductive Attributes

As it has been described above, in some scenarios an authorization decision should be based on another one taken in a different domain. Furthermore, there are specific situations where additional information is needed to take the authorization decision, that is, the deduction process cannot be based only on the authorization decision from the other domain, but some attributes should also be deduced. Therefore, the authorization decision received from the target domain should include these deductive attributes. This information depends specifically on the related authorization decision and, therefore, it cannot be recovered in the usual way, before requesting the PDP. For example, the end user could be allowed to utilize the SaaS service with a given resource, if the watermarking algorithm is provided by one of the additional services. Therefore the identifier of this watermarking service is queried from the company of the user and included as resource into the deductive request to the additional services.

The use of deductive attributes implies that an agreement should be established previously among the implied domains in order to define the set of attributes that can be deduced. Thus each domain should be able to obtain and to return these attributes if necessary. However, this is not an exclusive restriction for this scenario. The establishment of a common set of attributes is required in federated environments to allow interoperability among organizations, and several standard attribute schemes such as SCHAC [13] or eduPerson [6] can be found.

When these kind of attributes are required to take an authorization decision a reference has to be made in a similar way as it is done with the decisions. The elements ReferredDesignator and ReferredSelector shown in Listing 4 are extensions of the AttributeDesignator and AttributeSelector, respectively. In addition to the elements and attributes of the XACML definition they contain the domain which should provide the specific value. We assume that the XPATH expression in the AttributeSelector and thus the ReferredSelector may contain variables, which should refer to those defined in the scope of the policy set or policy.

After the deduction has been performed, the received deductive attribute can be used like a local attribute. Circular dependencies among the local definition of ReferredDesignators or ReferredSelectors should be detected and handled in the same way as those among “normal” variables.

4.4 Extension to the Framework

A generic framework for access control has been presented in [10, 11] and is widely used. One could argue that the deductive attributes could be resolved in the same way as other attributes. Also that the context handler could query different (including non local) PDPs for a single request. Nevertheless, in order to examine the impact of the above extension to the XACML language on the framework we have outlined two new elements in Figure 4, necessary from a deployment point of view.

The new elements are the Deductive Policy Information Point (DPIP) and the Deductive Policy Decision Point (DPDP). The former is resolving attributes at different domains referred to with the ReferredDesignator and ReferredSelector elements (see Section 4.3), the latter queries decision based on PolicyReference elements (see Section 4.2).

In either case if one of these elements is found during the evaluation of a policy, the handling is in accordance with the one of an unknown attribute. Thus the PDP replies to the Context Handler (CH) with a XACML Context response including the “NeedMoreInfo” decision and specifying the required information. The “NeedMoreInfo” is a new kind of response (additionally to Permit, Deny, Indeterminate and NotApplicable). The required information is the target domain and either the attribute that should be deducted or the respective decision. The PDP maintains the status of the evaluation process to complete it after the CH provides the required information. Attributes and variables from the Deduction are translated to the proper values in the XACML Context Response.

Based on the response from the PDP, the CH sends an authorization decision request to the DPDP respectively or an attribute query to the DPIP. The DPIP or DPDP discovers its proper corresponding entity in the other domain, where the query is fed to the related context handler. The
related request and response require the redefinition of existing elements as indicated in Figure 4. These elements and the corresponding types are shown in Listing 5. In addition to the existing elements they are extended by elements containing the requesting and targeted domain.

5. COMPLEXITY

The major observation on analyzing the complexity of the introduced deduction feature is that no changes regarding the general computation complexity compared to the existing XACML has taken place. In contrast to Datalog or other predicate logics the variables used as a reference to the domain should only have one unique value, according to the definitions in Sections 4.1 and 4.2. Thus the newly introduced deduction condition is evaluated once and the result will be included in the encapsulating expression. One might observe that with a careful implementation of the PDP, deduction requests could be avoided if local parts of a condition already predict the final result. As an example this deductive condition might be encapsulated in an and function. Assuming that another expression in this function could be evaluated locally to false, the request to deduce the decision at another PDP need not to be sent out at all. Regarding evaluation time, several deduction conditions to different domains could be evaluated in parallel.

In order to avoid any circular dependencies, which might lead to an undecidable request, the referred domains have to be specified in the definition of the policy. Guarding the reference to these domains, which are used for deducting decisions, does not add any complexity, as the list of referenced domains defines the potential space, and the consistency of the actual referred domain could be done in constant time. The actual check for circles due to the references to other domains is done while loading a policy, and not during the evaluation time.

Nevertheless requesting decisions from other domains consumes extra time due to the remote communication, but it does not add any complexity to the operation. One might observe that policy information points in the existing XACML standard might also get their actual data from remote entities. Thus controlling these time constraints during the evaluation of the policy might be one of the future challenges for XACML as well as for the introduced deductive policies.

6. EXAMPLE

In Section 1.1 we have sketched a potential application scenario. Deductive policies are useful in scenarios where it is necessary to coordinate the access control process among several domains, i.e. when an authorization decision in one domain depends on the authorization decision from another domain. As indicated in Figure 1, the scenario is based on four different domains, the SaaS Platform Provider, two different additional service providers and the company of the user, that reach an agreement to allow the employee of the latter to utilize the SaaS Platform including additional services. Besides, the agreement includes the condition on the type of watermarking that should be used.

Instead of the full XADM policy rule we give an overview of this policy Figure 5. As indicated, the example is realized with a policy set combining the results of two other policy sets with the permit-unless-deny algorithm (see [11, sec C.7] for details). The authoritative domain of the subject is determined with the help of a local function and stored in a variable. In the first policy set this authoritative domain is queried whether a general utilization is granted. Its result is combined with the local policies of the SaaS provider, assuming a specific policy set containing the local policies. Depending on the respective combining algorithm the local policies should be evaluated first by the PDP for performance reasons.

The second included policy set is used to check whether one of the embedded services grants the utilization of the watermarking algorithm. The identifier of this watermarking algorithm is queried via a ReferredSelector from the authoritative domain of the subject and stored in a variable as well. As we discussed in section 4.2, the mapping of Target among domains is solved by using ReferredAttributes. For instance, as Figure 5 shows, the definition of resource in the original authoritative domain is "access-resource" but the definition of resource in the referred authoritative domain is described as "watermarking". Note that this section of the example illustrates that it is necessary to change the attributes of a request. If at least one of the authoritative domains (the other one cut from Figure 5) of the embedded services permits an access this result is forwarded.

Figure 5: Structure of the example policy
7. CONCLUSION
In this paper we have presented a new approach on deduction of policy decision in a distributed environment such as SaaS. Our approach is based on an extension to the well known XACML enhancing it with the key feature of deducing policy decisions as well as attributes from different domains. Although, this concept is known from policy languages based on Datalog [7, 15, 16], XACML policies have a much larger usage in real systems. In addition we avoid the undecidability problem known in Datalog, due to the strict hierarchy implied in the specification, even though negation of deduced results is possible. Thus the deduction of access requests depending on the results of other policies, which are referring to different types of applications and/or are done by different domains responsible, could be easily modeled in a very similar fashion to today’s XACML policy condition. This will allow us to adopt existing tools to support the modeling and verification of these new type of policies. The reason for using existing and new policy combining algorithm is to integrate different results and the related obligations in a consistent way.

Additionally, due to the introduction of Authoritative Domains, a new level of abstraction has been introduced to the XACML based policies. The people responsible for a specific deductive policy do not have to consider, or even know, how the referred domain reached its decision, which they use for deduction in their policy. Thus, the analysis of their own policy becomes easier as these details do not have to be modelled and the policy itself should get smaller.

While remote requests to other domains add extra time for the evaluation, the complexity of this kind of policy is in the same class as XACML, as we avoid to search over a large search space in the as it is often the case in Datalog policies.

Depending on our ongoing work in our EU Project SWIFT and the experiences with these kind of policy rules, we might consider a further extension to support the evaluation of ranges of potential variable values, similar to Datalog. These experiences and findings will be the focus of upcoming publications.

8. REFERENCES

Obituary
The colleagues at University of Murcia, NEC Laboratories Europe and at the other partners of the SWIFT consortium were shocked to hear that Manuel Sánchez Cuenca passed away suddenly during the writting of the initial version of this paper. We will miss his friendly nature and always remember his contributions to the research community of Computer Science.