2011 Ninth Annual International Conference on Privacy, Security and Trust

Advantages of a Non-Technical XACML Notation in Role-Based Models

Bernard Stepien[1,2], Stan Matwin[1,2,3], Amy Felty[1,2]
[1]School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada
[2]Devera Logic Inc., Ottawa, Canada
[3]Institute for Computer Science, Polish Academy of Sciences, Warsaw, Poland
(bernard | stan | afelty)@site.uottawa.ca

Abstract— **As applications requiring access control and the environments in which they operate in become more complex, an acute need for better ways to manage access control rules has arisen. Decentralized access control, for example, requires sophisticated techniques for conflict detection and for managing rules across multiple applications with different rule formats. XACML is an OASIS standard whose interoperability qualities help in solving the latter problem. XACML has its own limitations, however. In particular, although it has the expressive power to specify very complex conditions like those needed in the ABAC (Attribute Based Access Control) model, users tend to avoid using its full power because of its verbosity. In this paper, we show how a non-technical notation we have proposed in our earlier work resolves this difficulty and allows users to work with a very compact and readable form of XACML rules, thus allowing them to take advantage of XACML's full expressive power. This expressive power can be exploited to write policies that are better organized. It can be easier, for example, to write a single possibly complex rule to cover a particular aspect of a policy as opposed to distributing the complexity over several rules with simpler conditions. As a result, policies are smaller, more compact, and easier to understand. Policy development becomes more manageable, allowing users to concentrate on the more central issue of choosing the model (RBAC, ABAC, PBAC or other) that is best suited to a particular application and policy. We show that using the full expressive power to better organize policies has a significant positive impact on PDP performance.**

Keywords- XACML, notation, access control, PDP performance.

I. MOTIVATION

A. Summary of role-based access control models

Specification languages for access control (AC) policies have evolved extensively in recent years. Early Access Control List (ACL) languages focused on the relation between users and resources. Including all associations between individual users and resources often resulted in policies with very large numbers of rules. Such policies are also very cumbersome to manage because of user status changes. A first solution to this administration problem was the introduction of Role Based AC (RBAC) [4][5] where permissions were allocated to roles. RBAC takes a two-step approach. The first step consists of assigning permissions to roles and the second consists of assigning roles to users. More importantly, the principle of inheritance between roles avoided duplication of definitions and separated duties to prevent users from being assigned mutually exclusive roles. The RBAC model has been considered to be inefficient for several reasons. First, differentiating roles in different contexts often proved to be difficult. This resulted in large quantities of role definitions—in some cases producing more roles than users. Second, RBAC remains somewhat coarse grained while modern requirements are increasingly fine grained. Finally, while the initial RBAC model was based on permissions only, the need to explicitly specify denial of access became unavoidable. These factors resulted in six variations of the RBAC model, defined in [11] and also variants of RBAC [7] and refinements at the privilege level [12]. Also, in [13], attempts are made to make RBAC parametric to consider contextual information.

The Attribute Based AC (ABAC) [28], [20] model is an evolution from RBAC that eliminates the user-role assignments and instead focuses on the attributes of a user required to grant access. One of those attributes can include the role a user has been assigned, thus inheriting from the RBAC model. ABAC is a very flexible model that is generally believed to be considerably easier to administer than RBAC. For example, the number of rules is reduced merely due to the absence of user/role assignment tables. ABAC has become a widely used model especially for application level AC such as web applications [17], [18] and database applications [22]. However, problems started to arise, due precisely to its ability to work at the application level. Different applications each have their own sets of AC policies. These can conflict with each other, or more importantly, conflict with general company policies. In addition, there is the burden of administering several sometimes overlapping policy sets. Legal implications in cases of damages resulting from faulty or conflicting AC rules led to legislation (SOX, HIPPA, etc.). The administration of disparate application level rules also proved to be very difficult, mostly due to ABAC's capability to specify complex rules. Evaluating such rules can be a difficult exercise in logic, especially for non-technical policy makers. In addition, the inherent decentralization of policy administration resulted in ontological problems such as the use of different names for the same attribute. This particular problem makes comparisons of policy sets for different

978-1-4577-0584-7/11/$26.00©2011 IEEE

applications even more difficult and requires the use of ontologies [15, 17, 21].

The Policy Based AC (PBAC) model is in principle not so different from ABAC. It too relies on attributes to determine access, but the main difference is that these attributes are defined at an enterprise level rather than at a local application level. The main motivation behind PBAC is to better ensure compliance to recent legislation. However, this creates another class of problems because the centralized rule attribute definitions may not match the local application definitions. This is a problem shared with ABAC, as mentioned above. Consequently, PBAC requires a complex architecture in order to ensure that AC requests by local applications can be understood by Policy Decision Points (PDPs) and appropriately grant or deny access.

Furthermore, such centralized enterprise-level AC is likely to be close to impossible in the case of dynamic environments, such as federated systems that use applications residing on the internet, where information is drawn from various unrelated business entities, known as the B2B model. These kinds of applications would normally be most appropriately served by the ABAC model, if it wasn't for the requirement to be adaptable to changing or even unpredictable environments. In such cases to avoid blocking the system, more complex mechanisms that evaluate risks must be used. This is the underlying principle of the Risk-Adaptive AC (RAdAC) model.

B. *Implementation of role-based models using XACML*

The reality is that all of the variants of the role-based AC models are used mostly because each of them is better adapted to specific applications as shown in the case study in [6]. This has resulted in commercially available tools that follow each of these models and use the XML based OASIS standard XACML policy description language [2] as a machine readable and interoperable representation.

Also, since the RBAC model is increasingly used for more fine grained AC applications with numerous additional attributes, the distinction between RBAC and ABAC is becoming less significant. Already, the combination of RBAC and ABAC was recommended in [8] in order to reduce the number of roles and rules.

All of the above described models, summarized in [14], also have increasingly taken advantage of the standardization of policy representation. Standardization ensures interoperability between different applications and also provides secure communication, for example over the internet. XACML has become a widely accepted solution because it has a rich data typing model and allows for the specification of complex conditions including the unrestricted use of disjunctions. However, XACML is very verbose and complex conditions expressed in XACML quickly become unreadable. This is particularly true when several policies need to be evaluated concurrently, as in the case of the RBAC model, where a policy can only be understood in the presence of other policies it inherits from. Our non-technical notation [1], which presents XACML conditions in a more natural language oriented way while still maintaining the precise structure of XACML, eliminates the problems associated with specifying complex conditions by the mere fact that they become more readable. As a result, it is possible to specify increasingly fine grained policies. As shown in our previous work [23], this capability has the potential to greatly reduce the number of rules and policies that have to be specified for a specific security application. The non-technical representation does not replace XACML. It merely hides the complexity of XACML syntax using a data model for attributes that ensures the correct translation in both directions, from the machine readable pure XACML to its human readable representation, and back again. In this paper, we show how the use of the non-technical notation can greatly improve the management of AC applications using any of the role-based models.

C. *Structural factors of role-based models*

One of the most important factors in choosing a particular role-based model is the number of policies required to describe an access control implementation. Migrating from RBAC to ABAC is usually considered a trade-off since RBAC potentially has 2^n combinations of roles while ABAC has a potential number of 2^n combinations of rules for n attributes, as shown in [8]. These numbers are the maximum number of combinations, and each application domain has its own realities. For example, in [6] it is reported in a banking environment case study that the number of roles represents around 3% of the number of users. However, we argue that a great portion of the excessive number of policies or rules of role-based models are the result of programming styles partly related to language rigidities rather than the underlying model itself.

The rest of our paper is structured as two main sections. Section II describes the issues of language rigidities and their consequences with respect to the number of policies required to describe an AC problem. Section III provides a detailed analysis of the benefits of the reduction of the number of policies and rules on performance of PDPs.

II. ADVANTAGES OF A NON-TECHNICAL NOTATION FOR ROLE-BASED MODELS

A. *Summary of the XACML non-technical notation*

The need for a non-technical notation for XACML was raised in OASIS XACML work groups in the early stages of the definition of the standard. XACML tool providers have attempted to address the problem, usually using fixed functionality GUIs that are notoriously inflexible, and thus defeat the fine grained policy advantage of using XACML. This use of GUIs has sometimes resulted in hard-coding the number and nature of attributes for the class of application that the tool is addressing. In recent years, the problem of the

technology gap between policy makers that define policies in plain natural language and policy implementers that code the policies in an implementation language such as XACML has reinforced this problem. The issues are very simple:

- Policy implementers may make errors in interpreting policy-maker defined policies, expressed most often in natural language.
- Policy implementers may make implementation errors of their own that are not verifiable by the policy makers who lack the technical knowledge to do so.

The above issues are illustrated in Fig. 1. They create a verification barrier for the policy maker, who normally is the only one that should be deciding what is correct in the implementation.

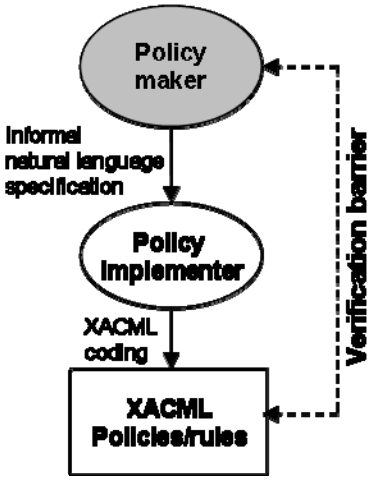

Figure 1. Technical knowledge gap

Our Non-Technical XACML notation [1] is based on three basic principles:

- Elimination of technical details around attribute names and values.
- Use of user-friendly operator names.
- Use of a horizontal tree representation as a structuring paradigm to define scope.

The most important aspect of the notation is that it is strictly a presentation notation and in no way a new language. The displayed policies and rules using this notation are the result of a direct translation from XACML using an attribute data model. The omitted XACML details are in no way eliminated; they are included in the internal representation that is not visible by the Policy Administration Point (PAP) tool user. The users are not performing any coding. Instead, they modify the policy text indirectly by clicking on buttons that perform various functionalities such as adding or deleting sub-constraints. Both attributes and values are selected using GUIs built from check-boxes. The data model defines the data type and the nature of attributes (subject, resource, action, environment, etc.). The use of attribute data models is very common in PAP tools. However, we go a step further by using them to completely hide technical details from the user. The data model is built by technical engineers as the first part of the PAP development task, as shown on Fig. 2. Thus, we achieve a separation of concerns.

Our use of GUIs is different from most other PAP tools. Like other tools, we use a GUI to select attributes and values, but not to display them. Our non-technical notation allows the display of the entire rule condition or target at once in a readable natural language text, thus providing a succinct representation of a policy's logic

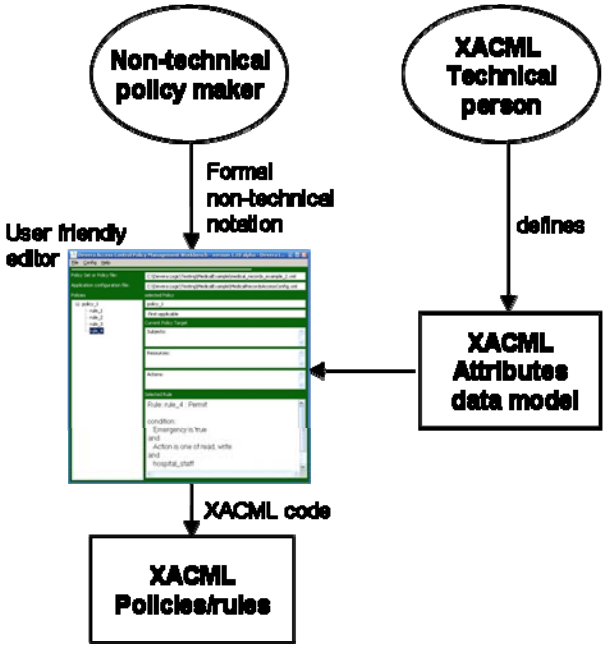

Figure 2. Distribution of knowledge

For example, in order to express that a nurse can write a diagnostic only in an emergency situation or during weekends while a physician can do so only between 9:00 to 5:00 can be expressed with this non-technical notation using a rule with a complex condition as follows:

```
Rule read_write_medical_record  effect: Permit

   Action is write
and
   Medical_record is diagnostic
and
   Role is one of
      nurse
            provided that Emergency is true
         or
            DayOfTheWeek is one of Saturday, Sunday
      physician
         provided that time is between 9 and 17
```

Due to its compactness, the notation enables AC policy administrators to focus on logic content rather than on language syntax. We have found that what remains of technicality, i.e. the tree representation, is easy to learn by a non-technical user. The compactness of the notation has two

benefits: first it reduces the amount of information an administrator has to process, and second it also enables viewing of several related policies at the same time in a single window.

B. Rigidity of the XACML target concept

1) Readability of target elements

In XACML targets, normally, it is possible to have several matches on a target element (subject, resource and action). One problematic issue with targets is that they are hard to read because there is no explicit indication about whether the multiple matching specifications are linked via conjunction or disjunction operators. One has to remember that the XACML standard specifies that multiple target elements are considered disjunctions, while multiple element matches (resourceMatch, etc.) are considered as conjunctions. Here, the non-technical notation merely displays the implicit conjunction and disjunction operators explicitly. For example, consider the XACML target that expresses a policy that two different pieces of equipment (resources) can be used only on specific days.

XACML target:

```
<resources>
  <resource>
    <ResourceMatch MatchId=string-equal>
       <AttributeValue …>scanner</…>
       <ResourceAttributeDesignator
            AttributeId="…:resource-id"/>
    </ResourceMatch>
    <ResourceMatch MatchId=string-equal>
       <AttributeValue …>Monday</…>
       <ResourceAttributeDesignator
            AttributeId="…:DayOfTheWeek"/>
    </ResourceMatch>
  </resource>
  <resource>
    <ResourceMatch MatchId=string-equal>
       <AttributeValue …>computer</…>
       <ResourceAttributeDesignator
            AttributeId="…:resource-id"/>
    </ResourceMatch>
    <ResourceMatch MatchId=string-equal>
       <AttributeValue …>Thursday</…>
       <ResourceAttributeDesignator
            AttributeId="…:DayOfTheWeek"/>
    </ResourceMatch>
  </resource>
<resources>
```

The above XACML code is displayed using the non-technical notation as follows:

Equivalent non-technical notation representation:

```
Resources:
      Resource-id is scanner
   and
      DayOfTheWeek is Monday
or
      Resource-id is computer
   and
      DayOfTheWeek is Thursday
```

2) Disadvantages of the target conjunctive model

The second issue with XACML targets is the limitation in expressive power due to the fact that subjects, resources and actions are related implicitly by a conjunction operator. This prevents writing logical expressions where various combinations of subjects, resources and actions could be specified in a single policy or rule. For example two different resources r_1 and r_2 that would be associated respectively with action a_1 for r_1 and a_2 for r_2 would require two separate policies or rules to be specified using an ordinary XACML target. If, on the other hand, the target could be written using a single logical expression mixing subjects, resources and actions using disjunction operators, it would require only one policy as shown in Fig. 3a.

Target:	Target:
r_1 and a_1 or r_2 and a_2	r_1 and a_1 and C_1 or r_2 and a_2 and C_2
Figure 3a: target element combinations	**Figure 3b: target condition combinations**

3) Disadvantages of the rule target/condition conjunctive model

A third issue with XACML targets is similar to the second but with the additional problem of a rule target's relation with the rule's condition part. The condition part applies to the entire target either at the rule, policy or policy set level and thus here again multiple policies or rules must be specified with individual combinations of conditions for the target elements. This has the immediate result of forcing the administrator to write separate policies for each combination of targets and conditions, while the same could have been represented in a single rule or policy if the logic of target elements could be related to the condition using a single logical expression, as shown in Fig. 3b. In fact, the distinction between resources and subjects has been questioned in [16] where it has been found that attributes are sometimes interchanged in federated systems. The same problem occurs with the XACML RBAC profile [3] where roles appear as subjects in the user/role assignments in policy definitions, while they appear as resources in the role permission assignment part of policies. Thus, in order to alleviate the above problems, we strongly recommend the use of single logical expressions that allow the combination of target elements and conditions. We show that this has the direct result of reducing the number of policies or rules, thus making the administration of policies considerably easier. The only associated problem with this proposal is the use of indexing to increase PDP performance. We discuss this topic in section III, as interesting solutions exist. Using single logical expressions that combine both targets and conditions allows one to greatly reduce the number of policies required when specifying a given access control application. This can considerably improve the implementation of the RBAC model in particular.

C. Scattering of information in RBAC models

In the XACML RBAC profile, role inheritance is represented using the *PolicySetIdReference* and *PolicyIdReference* language elements. While inheritance is an important re-

usability technique, it has the undesirable side effect of dispersing information. This dispersal is by definition unavoidable. However, we argue that since our non-technical notation already makes the reading of one single policy or rule easier, it is of an additional advantage when having to view several policies specifying several inherited roles at the same time. This is simply because of the fact that when the quantity of reading material increases, going through verbose XML tags further complicates the understanding process. Thus, it is even more essential to have features in XACML PAP editors that allow browsing through policy sets and policy references by synthesizing the logic of the policies with the logic of the inherited policies, displaying all of the relevant information in one place. To show the benefits of our technique, consider the example definitions of the manager and employee roles shown in [3].

Original separate policies:
Employee role definition:

```
      resource is purchase_order
   and
      action is create
```

Manager role definition:

```
      resource is purchase_order
   and
      action is sign
```

PolicySetIdReference: PPS:employee:role

This example that illustrates that a manager inherits from an employee also shows how the XACML RBAC profile inheritance can lead to duplication of information: using complex conditions can also make the understanding of the net effect of the inheritance difficult. XACML inheritance is somewhat different from traditional OO inheritance mechanisms since in OO programming, inheritance is used to extend a class in terms of attributes and methods. Instead, XACML inheritance is about inheriting information, namely policy logic, in definitions. One problem that arises is that these logics may overlap. There is no mechanism such as semantic checking via language compilers to detect this problem. In the XACML RBAC profile, both the manager role and the inherited employee role share the common resource specification of purchase order. This information is distributed over the two definitions. Using the techniques we have described, we can synthesize this distributed information and eliminate the duplication by factoring out common expressions as follows, thus displaying the net result of the inheritance to enable the reader to assess the full effect of an inheritance:

```
      resource is purchase_order
   and
      action is one of create, sign
```

Again, displaying the synthesis of inherited logic doesn't imply coding. The user is still expected to compose the policies separately. The dilemma of document-readability and expressiveness of RBAC models represented as XML documents has already been discussed in [10], but that discussion concentrates on the nesting of tags. They too propose a tree structure derived from the one that XML already provides for a role graph bank database application.

From the above discussion, we recommend merging target elements (subject/resource/action) in a single logical expression and we also recommend the merging of rule targets and conditions. This would add flexibility to the flexibility resulting from the addition of attributes to the RBAC model as already recommended in [8].

D. The need for a XACML rule inheritance mechanism

Finally, using complex conditions in rules may not be applicable to the RBAC model because the RBAC profile for XACML [3] uses both policy set and policy ID references to implement role inheritance. XACML rules currently do not have such an inheritance mechanism, even in version 3.0. We thus propose an extension of the policy inheritance mechanism to rules in XACML to allow the full benefit of the complex conditions with the RBAC model. This could be implemented with a *RuleIdReference* tag. This also raises the problem of the differentiation between policies and rules. Traditional AC languages are ambiguous in this respect. In the case of XACML rule inheritance, the problem is to determine if a rule of a given policy can inherit from a rule of another policy. This problem is for further study. We also argue that a rule inheritance mechanism could considerably reduce the complexity of the XACML RBAC profile [3]. We believe that in its current form it is not usable by a non-technical user in the first place, whether a non-technical notation is used or not.

III. IMPACT OF POLICY STRUCTURE ON PDP PERFORMANCE

The policy programming style we have presented in the previous section has the net result of enabling a reduction in the number of policies or rules required in order to specify a given AC problem. But fewer policies or rules also has a positive impact on performance.

PDP performance is a widely researched subject especially for complex systems using XACML. However, most of this research concentrates on PDP implementation programming styles. Here we focus instead on the programming style of the policy sets before they are loaded into a PDP for execution.

A. Principal causes of bad PDP performance

Performance of PDPs has been studied in detail in [25], [26]. These studies focus on the comparison of the performance of various available open source PDP implementations using randomly generated policy sets. They also have compared the impact of policy set structure in terms of policies and rules, strictly implementing logic at the target level with subjects, resources and actions specifications. Their conclusion was that performance of open source PDPs in general is bad. In [26],

more efficient PDP implementations are proposed. In [23] we have shown that reducing the number of policies or rules by using more complex conditions in rules considerably improves the performance of conflict detection algorithms implemented in Prolog, in particular those that compare policies and rules against each other. Large numbers of policies are the direct result of programming styles that use only conjunction operations in logic to specify a policy. One of the reasons for this, as we have discussed earlier, is that the XACML language itself limits the administrator to the use of conjunctions when using XACML targets exclusively. This phenomenon is widespread and can be found even in the formal definition of policies found in [18] where only conjunctions are used.

B. *Using predictable performance generated policy sets*

We also use an approach to PDP performance measurement on generated policy sets. However, instead of using random generation as in [25], we incorporate information that is based on known combinations of values for attributes defined in our application data model. We also include requests that can only be matched by the policy and rule located at the bottom of the policy set under a combining rule algorithm that uses the first applicable rule to determine if a request is granted or denied. This forces the PDP to at worst go through the entire set of policies, though in practice may depend on the search implementation used and whether or not indexing is used. This makes the problem more predictable and easier to measure. In addition to the predictability of performance, this approach in fact also corresponds to the most common policy programming style, in particular the subject/resource/action conjunctive model that we have mentioned above. However we do not rule out random generation of policies because repeated random generation has its own virtues. They produce unexpected structure in policy sets that could reveal different classes of PDP performance problems.

In our previous work [23] studying performance of our conflict detection algorithm, the differences of performance were mostly attributed to the intelligent back-tracking capabilities of Prolog, in addition to the quadratic nature of (n x n)/2 -n rules comparisons in general. In this research, we have focused on the performance of PDPs using the same approach to structuring policy sets, namely the use of more complex conditions in rules to reduce the required number of policies or rules. In this experiment, we only compare a request to the various policies, which corresponds to a one-to-many instead of a many-to-many comparison. Thus, the quadratic nature of the conflict detection problem does not exist when measuring PDP performance. As we had shown in previous work, using disjunctions on attribute values, we can reduce any number of single policies that represent one combination of attribute values at a time to only a single policy and rule. This results in an extreme reduction in the number of rules and is only for demonstration and measurement purposes. In real policy sets, the reduction will likely be to the number of rules on subsets rather the n-to-1 reduction of our systematically generated example. A transformation algorithm to reduce the number of policies or rules of such policy sets is work in progress.

C. *Choice of stylistic strategies and performance*

In addition to the issue of representation of conditions, we have also compared the performance between a policy set where the logic is located either entirely in a policy target with one where it is entirely in a rule condition. This approach was guided mostly by informal discussions with PDP implementers that indicated that policy targets were indexed while rules conditions were not. Our experimental policy generation method resulted in two cases: one case has many policies each containing a target and a single catch-all rule with no condition; the other case has a single policy that contains many rules with no targets but instead only conditions. We created an application with 4 attributes each having respectively 7,8,6 and 7 values that results in 7x8x6x7 = 2352 combinations of such cases with either 2352 policies containing a single rule without a condition or a single policy containing 2352 rules each containing a complex condition.

D. *Performance measurement results*

The results showed that using rule conditions exclusively requires on average double the time to evaluate requests than the policies using targets exclusively. At first glance, these results could favor the policy target approach that is widely used in industry over the all-rule-condition approach that we advocate. The target approach also supports the use of efficient table indexing mechanisms. However, in an experiment using the above recommended policy consolidations, we observed that the reduction in the number of rules produces a considerable gain in performance of the open source PDP (sunxacml [27]). These gains are by far outperforming the gains associated with indexing. This gain is comparable to that achieved with the conflict detection algorithm in our previous work. The request evaluation time was consistently below one millisecond while all other approaches were above 100 ms on average for the best case. Thus, it is clear that stylistic considerations in writing policies can have a significant impact on PDP performances in addition to making policies more manageable from an administration point of view. This result is actually surprising, because both experiments use a request that can evaluate to a permit effect only with the last policy in the policy set. This means that a comparison with each value of the value set of an attribute has to be performed; one would normally expect the target approach to perform better because of indexing.

The reason could be the same as for the conflict detection experiments. Using numerous policies representing individual combinations, the matching mechanism has to repeatedly re-match all of the attributes composing a policy target. For example, if a policy has n attributes $a_{1, \ldots, a_n}$ with $n_1, \ldots, n_n$ possible values for each attribute, a total worst number of n_1 x … x n_n x n matches will be required for each of the individual target based policies, in order to get to the last policy, the one that permits the request. With a single policy and rule

containing the complex condition using disjunctions between all the values, only $n_1 + \ldots + n_n$ matches are required for the same request. It is to be noted that in the first case, the number of matches is not linear in relation to the number of attributes and quickly runs into combinatorial problems. Consider a concrete example with 3 attributes consisting respectively of 10, 5 and 4 possible attribute values. This would require 10 x 5 x 4 x 2 = 400 matches to be performed for the policy that uses targets compared with only 10 + 5 + 4 = 19 matches with the single rule that uses a complex condition. In fact, Prolog backtracking, that was believed to be the prime factor in the good performance of the conflict detection algorithm for policies with single rules, behaves in a similar fashion when comparing values in the current experiments, where the number of matches is additive rather than multiplicative. Again, these numbers indicate the worst case only, where each attribute is always evaluated against all its values. Normally this should not be the case since in good programming practices the first entirely mismatching attribute should stop any matching attempts on the remaining attributes involved in a conjunction. Also, it is to be noted that there still remains a random factor in this experiment since the actual matching of values is also influenced by the characters composing these values. These are by definition unpredictable. Thus our measurements are based only on the number of values rather than their composition.

E. A performance evaluation example

The following example is applicable whether the individual rules are expressed in terms of XACML targets or rule conditions. The example uses 3 attributes with respective number of values {3,3,2}. The matching of the request against *policy_18* below will require 30 comparisons using the most efficient approach, which compares attributes only if the previous attribute has matched. However, some PDPs are particularly inefficient and systematically compare every attribute for each policy. This would represent the worst case scenario, which in this example would require 3 x 3 x 2 x 3 = 54 comparisons.

In the following example, it is clear that the attributes X, Y and Z have to be evaluated for each policy over and over again. For example each attribute must be evaluated 18 times.

Request:

Attribute	Value
X	30
Y	"c"
Z	false

Individual value combinations policies:

```
Policy_1: X is 10 /\ Y is "a" /\ Z is true
Policy_2: X is 10 /\ Y is "a" /\ Z is false
Policy_3: X is 10 /\ Y is "b" /\ Z is true
Policy_4: X is 10 /\ Y is "b" /\ Z is false
Policy_5: X is 10 /\ Y is "c" /\ Z is true
Policy_6: X is 10 /\ Y is "c" /\ Z is false
Policy_7: X is 20 /\ Y is "a" /\ Z is true
Policy_8: X is 20 /\ Y is "a" /\ Z is false
Policy_9: X is 20 /\ Y is "b" /\ Z is true
Policy_10: X is 20 /\ Y is "b" /\ Z is false
Policy_11: X is 20 /\ Y is "c" /\ Z is true
Policy_12: X is 20 /\ Y is "c" /\ Z is false
Policy_13: X is 30 /\ Y is "a" /\ Z is true
Policy_14: X is 30 /\ Y is "a" /\ Z is false
Policy_15: X is 30 /\ Y is "b" /\ Z is true
Policy_16: X is 30 /\ Y is "b" /\ Z is false
Policy_17: X is 30 /\ Y is "c" /\ Z is true
Policy_18: X is 30 /\ Y is "c" /\ Z is false
```

Equivalent single policy with disjunctions:

With the version of the above policy expressed with a single rule (see below), the policy attributes are matched after only 8 comparisons of attributes to produce an authorization. This represents approximately four times less computation than with the conjunction oriented version above using the most efficient evaluation scheme, and seven times less for the most inefficient evaluation approach. In the following example, it is clear that each attribute needs to be evaluated as many times as its number of values. Here, X and Y are evaluated only three times while Z is evaluated only twice.

```
   X is one of 10, 20, 30
and
   Y is one of "a", "b", "c"
and
   Z is one of true, false
```

The PDP performance experiments we have seen so far were based on standard XACML data typing. More complex user defined data types [24] can also affect PDP performance. However, all of these measurements have been achieved on relatively inefficient open source versions of PDPs. The service oriented architecture (SOA) PDP presented in [19] produces request evaluation time below 1 ms on large policy sets. We provided our predictable large sets (2352 policies) to them, along with the corresponding single rule example, and they measured a similar gain in performance between the two approaches using their SOA based PDP.

In addition to the above experiments, we have used our Prolog CLP representation of XACML policies to create a PDP, in order to compare it to the Java implementation of sunxacml, and find out if Prolog's default internal indexing provided any sort of advantage. We have measured a gain of 50% in evaluation time compared to sunxacml with no significant difference for the two target vs. condition approaches. Although this is an improvement, we do not consider it as significant enough to be noteworthy. It appears that reducing the number of policies and rules through the use of complex conditions is considerably more advantageous. The improvement in performance increases significantly with the complexity of the case. While the best gain can be exponential, the average expected speed-up needs to be investigated.

However, the use of very complex conditions like our extreme case is difficult using traditional PAPs that use XACML directly because of its verbosity. In contrast, the non-technical notation allows such complex conditions to be specified

without any significant effort. Thus, such a non-technical notation is an essential component of the success of the role-based models approach.

CONCLUSION

The non-technical XACML presentation notation makes the use of complex conditions in policy rules practical. This enables a completely different AC policy programming style based on complex conditions that has the potential to produce a significant reduction in the number of policy rules required for a given access control application problem. As a result, following the role-based models to define policies becomes more manageable and less prone to error, and also improves PDP performance. In order to improve the RBAC model we recommend the creation of an inheritance mechanism at the rule level in addition to the existing policy and policy set reference.

Acknowledgement. The authors are grateful to NSERC and the Ontario Centres of Excellence for their support for this research.

REFERENCES

[1] B. Stepien, A. Felty, and S. Matwin, A non-technical user-oriented display notation for XACML conditions, E-Technologies: Innovation in an Open World, in *Proc. of the 4th International MCeTech Conference*, Springer LNBIP 26, 2009.

[2] XACML, OASIS standard, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[3] XACML Profile for Role Based Access Control (RBAC), 2004, http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf.

[4] D. F. Ferraiolo, D. R. Kuhn, Role-Based Access Control, in *Proc. of the 15th National Computer Security Conference*, pages 554-563,1992.

[5] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, Role-Based Access Control Models, in *IEEE Computer,* 29(2):38-47, 1996.

[6] A. Schaad, J. Moffett, J. Jacob, The Role-Based Access Control System of a European Bank: A Case Study and Discussion, in *Proc. of the 6th ACM Symposium on Access Control Models and Technologies*, pages 3-9, 2001.

[7] A. Lin, Integrating Policy-Driven Role Based Access Control With The Common Data Security Architecture, Technical Report, HP Laboratories, HPL-1999-59, 1999.

[8] R. Kuhn, E. J. Coyne, T. R. Weil, Adding Attributes to Role-Based Access Control, in *IEEE Computer*, 43(6):79-81, 2010.

[9] D. Ferraiolo, R. Kuhn, R. Sandhu, RBAC Standard Rationale, in *IEEE Security & Privacy*, 5(6):51-53, 2007.

[10] R. Chandramouli, Application of XML Tools for Enterprise-Wide RBAC Implementation Tasks, in *Proc. of the 5th ACM Workshop on Role-Based Access Control*, 2000.

[11] S. Barker, P. J. Stuckey, Flexible Access Control Policy Specification with Constraint Logic Programming, in *ACM Transactions on Information and System Security*, 6(4):501-546, 2003.

[12] M.A.C.Dekker, J. Cerderquist, J. Crampton, S. Etalle, Administrative Privileges in RBAC, in *!st Benelux Workshop on Information and System Security*, 2006.

[13] J.K. Adams, A Service-Centric Approach to a Parameterized RBAC Service, in *Proc. of the 5th WSEAS International Conference on Applied Computer Science*, 2006.

[14] A survey of Access Control Models, workshop material of *Privilege (Access) Management Workshop*, http://csrc.nist.gov/news_events/privilege-management-workshop/PvM-Model-Survey-Aug26-2009.pdf, 2009.

[15] T. Priebe, W. Dobmeier, C. Schläger, N. Kamprath, Supporting Attribute-based Access Control in Authorization and Authentication Infrastructures with Ontologies, in *Journal of Software*, 2(1):27038, 2007.

[16] T. Priebe, W. Dobmeier, B. Muschall, G. Pernul, ABAC – Ein Referenzmodell für attributebasierte Zugriffskontrolle, in *Proc. of Sicherheit*, 2005.

[17] G. Bayer, D. Sengupta, T. Wang, A. Vogt, PrplAc: Attribute-based Access Control in PRPL for Fine Grained Information Sharing using Semantic Web, Technical Report, http://senguptas.org/Documents/cs343-PrplAc.pdf.

[18] H. Shen, F. Hong, An Attribute-Based Access Control Model for Web Services, in *Proc. of the 7th International Conference on Parallel and Distributed Computing,* Applications and Technologies, pages 74-79, 2006.

[19] M. Cheaito, R. Laborde, F. Barrère, A. Benzekri, A deployment framework for self-contained policies, in *Proc. of the 6th International Conference on Network and Service Management*, 2010.

[20] L.Wang, D. Wijesekera, S. Jajodia, A Logic-based Framework for Attribute based Access Control, in *Proc. of the 2004 ACM Workshop on Formal Methods in Security Engineering*, pages 45-55, 2004.

[21] C. A. Ardagna, S. De Capitani di Vimercati, S. Paraboschi, E. Pedrini, P. Samarati, An XACML-Based Privacy-Centered Access Control System, in *Proc. of the 1st ACM Workshop on Information Security Governance*, pages 49-58, 2009.

[22] Sonia Jahid, Imranul Hoque, Hamed Okhravi, Carl A. Gunter, Enhancing Database Access Control with XACML Policy, poster at *16th ACM Conference on Computer and Communications Security*, http://www.cs.illinois.edu/homes/sjahid2/pub/myabdac-ccs09-abstract-JahidHOG.pdf, 2009.

[23] B.Stepien, A. Felty, S. Matwin, Strategies for Reducing Risks of Inconsistencies in Access Control Policies, in *Proc. of 5th International Conference on Availability, Reliability, and Security*, pages 140-147, 2010.

[24] M. Cheaito, R. Laborde, F. Barrère, A. Benzekri, An extensible XACML authorization decision engine for context aware applications, in *Joint Conferences on Pervasive Computing*, pages 377-382, 2009 .

[25] F. Turkmen, B. Crispo, Performance Evaluation of XACML PDP Implementations, in *Proc. of the 2008 ACM Workshop on Secure Web Services*, p. 37-44, 2008.

[26] B. Butler, B. Jennings, D. Botvich, XACML Policy Performance Evaluation Using a FlexibleLoad Testing Framework, in *Proc. of the 17th ACM Conference on Computer and Communications Security*, pages 648-650, 2010.

[27] SunXACML, url: http://sunxacml.sourceforge.net/, last access February 2010.

[28] H. Shen, F. Hong, An Attribute-Based Access Control Model for Web Services, in *Proc. of the 7th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 74-79, 2006.