A Non-Technical XACML Target Editor for Dynamic Access Control Systems

Bernard Stepien, Amy Felty
University of Ottawa
School of Electrical Engineering and Computer Science
Ottawa, ON Canada
(bernard | afelty)@eecs.uottawa.ca

Stan Matwin
Dalhousie University, Faculty of Computer Science
Canada Institute for Computer Science,
Polish Academy of Sciences
Halifax, NS Canada
stan@cs.dal.ca

Abstract— **XACML is a powerful and flexible access control (AC) policy language. It is an OASIS standard that is now widely used in a variety of applications, particularly those that require inter-operability between AC systems. The language definition includes a precise grammar, syntax, and semantics, and it is both expressive and verbose. This combination of expressive power and verbosity can lead to difficulty in understanding the language's syntax and semantics for both technical and non-technical users alike. As a result, reducing the difficulty of editing XACML policies has become an intense area of research. In our own work in this area, we previously showed how to render complex XACML conditions using a non-technical display notation and showed that it is easy to use this notation with interactive plain text editors that do not require any technical coding. Although XACML conditions are expressive and flexible, XACML targets are actually the most commonly used XACML language construct. They have an additional level of complexity, especially in version 3.0, due to the fact that the form and kinds of XACML constructs allowed in targets is much more limited. This paper extends our previous work, showing how the same powerful and flexible interactive editing principles can be applied to targets in order to allow users to use natural logic rather than implementation logic. We extend these principles and fully integrate them into our editing tool, easyXACML. This tool is usable by users with no technical knowledge of XACML, thus making XACML totally transparent to the user, while still retaining all of its functionalities and semantics. Our tool thus allows users to focus on policy logic rather than on details of syntax. As a result, the risk of errors in policies is greatly reduced.**

Keywords-component; Access control, XACML, policy administration point, ABAC, RBAC.

I. MOTIVATION

The XACML language [1][2] is based on XML and has a rich typing system and set of access control (AC) logic specification constructs. These features make it very efficient for specifying the most recent AC models, especially those that are particularly efficient for expressing fine grained AC, such as the ABAC [3] and RBAC [4][5] models, as well as many other derived models. XACML has one major advantage over text based specification languages in that it eliminates the cost of developing dedicated compilers. It does so by making use of a large body of open source generic XML tools in combination with the XACML schema. It is widely used mostly because as a standard, it fulfills the needs for inter-operability capabilities [6], which is a basic requirement for federated systems.

However, XACML is a complex language with lots of constructs and despite XML's self-describing markup approach, many factors make it difficult to use off-the-shelf generic XML tools for implementing a policy administration point (PAP) efficiently. Such factors include the length of these self-describing XML tags, long domain names, a vast collection of operators, including user defined operators, and structural components. One outcome is that users often avoid XACML altogether and instead use more traditional approaches, such as relational databases using SQL as in the medical application discussed in [7]. As a result, there have been many attempts to alleviate this problem and they can be classified into two broad categories:

- PAPs for technical programmers
- PAPs for non-technical users

In a fully working implementation in previous work [8], our contribution was to provide a solution for non-technical users for XACML rule conditions, which consist of plain Boolean expressions. Also, in previous work [9], we have shown how to represent XACML targets using the same notation as for XACML rule conditions. The editing capabilities provided by easyXACML are particularly useful in domains where dynamic access control is important. For example, medical applications, military applications, and emergency response team applications are domains where rules change often. It is particularly important that changes can be made quickly by medical, military, or emergency personnel, who are unlikely to be XACML experts. However, while this approach was developed primarily for non-technical users, it turned out that technical AC administrators equally benefit, mostly because they can focus on AC logic rather than on syntax.

Finally, another major advantage of the non-technical approach is that it allows a completely different style of composing policies. The prevailing style is to specify simple

AC logic involving only conjunctions of single instances of logic criteria for a given attribute and to rely on the natural hierarchical nature of the XACML language to structure policy sets into policies and rules. The two main language constructs for specifying AC logic in XACML are the multi-level (policy set, policy, and rule) target and the rule-specific condition. The main programming style currently used by the vast majority of XACML users consists of locating the essential part of AC logic in targets, and using rule conditions only for minor refinements.

In XACML 2.0, another restrictive practice was to confine AC logic to single combinations of subjects, actions, resources and environment attributes. Early attempts to use the full flexibility of rule conditions encountered limited enthusiasm. Also, the XACML target has the advantage that it orders policy logic into predictable categories, thus making it efficient to evaluate by policy decision points (PDP). However, this advantage is somewhat lost by duplication of information that results from using targets. In previous work, we have shown the structural advantages of using the expressive power of conditions to represent logic because they are closer to the Boolean expression representation, and while there is no trace of other publications on this matter, we have found at least a patent for an algorithm to transform XACML targets into rule conditions [10], which seems to indicate that the problem is well known. The main reason invoked for the lack of success of the rule condition approach was the difficulty of representing complex logic in databases. While rule conditions allow very complex expressions, the targets actually do also allow complex expressions in a limited way. This is due to the fact that targets can be used to specify alternate and supplementary matches. This has led to solutions that transform complex XACML targets into access control lists (ACL) as in [11]. On the other hand, others try to do exactly the opposite [12], transforming ACL to XACML. However the latest version of the standard, XACML 3.0, represents a breakthrough because it provides the capability for more complex logic in XACML targets by allowing the mixing of subject, action, and resource matches. This capability also enables the policy writer to reduce the number of individual policies and rules required for a given AC application. However, the only drawback of this approach is that the depth of the logic trees for a given target is limited to three. Deeper logical expressions can be implemented by distributing logic among the XACML hierarchical structural elements (policy set, policy and rule targets) but always in a somewhat restricted manner due to the implicit conjunction between policy structural elements in the hierarchy.

XACML conditions use straightforward logic structured naturally using explicit conjunction and disjunction operators, which can be mapped on a one-to-one basis to non-technical terminology. XACML targets, on the other hand, use implicit conjunctions and disjunctions, either totally or partially depending on the XACML version but also are based on shallow depth logic expressions. This is a significant restriction. These restrictions force the users to use relatively inefficient programming styles, one of which is code duplication. While XACML 3.0 supersedes the 2.0 version, there is still a wide community of users that use version 2.0 and, more importantly, is reluctant to migrate, mostly because they find version 3.0 too complex. Also, XACML target expressions require some duplication in definitions that is unnecessary in a non-technical representation. Thus, our contribution here is to show how complex duplicate XACML definitions can be made transparent to the user when editing a XACML target. This extends previous work that addressed the complexity of Boolean expressions represented by XACML rule conditions [8]. All other aspects of XACML policies and policy sets are covered in previous work and not shown here.

II. XACML TARGET CHALLENGES

The main difference between XACML targets and XACML rule conditions is that targets have the following limitations depending on the XACML version:

- In XACML 2.0, subjects, resources and actions (categories) cannot be mixed in matches. They are described in separate sections for each category and these sections are related only via implicit conjunctions.
- In XACML 3.0, subjects, resources and actions can be mixed in matches.
- In both XACML versions, each category can be described in terms of alternate groups of matches.
- In both XACML versions, each alternative group can contain only a conjunction of matches.

Thus, while the depth of rule conditions is unlimited, the depth of target expressions is limited to 3 and with a very specific hierarchical structure in terms of conjunction and disjunction operations allowed at each level as shown in Figure 1.

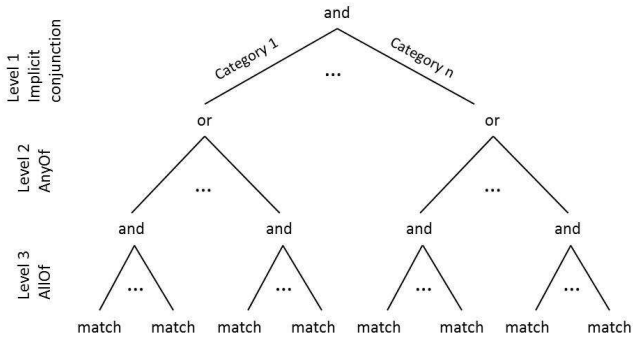


Figure 1: XACML target restrictions

Thus, the challenge consists of introducing these restrictions in the tool but also by hiding them from the non-technical user. This can be easily achieved by using context driven principles that allow presenting only what is allowed at

a given step of a target construction task. Aside from the capability of mixing categories in XACML 3.0 targets, the difference between the two versions is only syntactical and is thus irrelevant in the non-technical representation editing phase. Also, from a syntax point of view, while in version 2.0 all of the logical operators shown in Figure 1 are implicit, in version 3.0 only levels 2 and 3 are explicitly implemented using *AnyOf* and *AllOf* tags, level 1 remaining implicit. For example, a policy that specifies that physicians can read diagnosis or surgery reports without restrictions while nurses can do so only while in an emergency or operating room would be specified as a policy target in XACML 3.0 as follows:

```
01 <Target>
02   <AnyOf>
03     <AllOf>
04       <Match MatchId= "string-equal">
05        <AttributeDesignator
06          Category="subject-category"
07          AttributeId="subject-id"
08          DataType="string"/>
09        <AttributeValue DataType="string"
10                    >physician</AttributeValue>
11       </Match>
12     </AllOf>
13     <AllOf>
14       <Match MatchId= "string-equal">
15        <AttributeDesignator
16          Category="subject-category"
17          AttributeId="subject-id"
18          DataType="string"/>
19        <AttributeValue DataType="string"
20                    >nurse</AttributeValue>
21       </Match>
22       <Match MatchId= "string-equal">
23        <AttributeDesignator
24          Category="subject-category"
25          AttributeId="Location"
26          DataType="string"/>
27        <AttributeValue DataType="string"
28              >emergency room</AttributeValue>
29       </Match>
30     </AllOf>
31     <AllOf>
32      <Match MatchId= "string-equal">
33       <AttributeDesignator
34         Category="subject-category"
35         AttributeId="subject-id"
36         DataType="string"/>
37       <AttributeValue DataType="string"
38                     >nurse</AttributeValue>
39      </Match>
40      <Match MatchId= "string-equal">
41       <AttributeDesignator
42         Category="subject-category"
43         AttributeId="Location"
44         DataType="string"/>
45        <AttributeValue DataType="string"
46             >operating room</AttributeValue>
47     </Match>
48    </AllOf>
49   </AnyOf>
50   <AnyOf>
51    <AllOf>
52     <Match MatchId= "string-equal">
53      <AttributeDesignator
54        Category="resource-category"
55        AttributeId="resource:resource-id"
56          DataType="string"/>
57       <AttributeValue DataType="string"
58             >surgeries report</AttributeValue>
59      </Match>
60     </AllOf>
61     <AllOf>
62       <Match MatchId= "string-equal">
63        <AttributeDesignator
64          Category="resource-category"
65          AttributeId="resource:resource-id"
66          DataType="string"/>
67        <AttributeValue DataType="string"
68                   >diagnosis</AttributeValue>
69       </Match>
70     </AllOf>
71    </AnyOf>
72    <AnyOf>
73     <AllOf>
74       <Match MatchId= "string-equal">
75        <AttributeDesignator
76          Category="action-category"
77          AttributeId="action:action-id"
78          DataType="string"/>
79        <AttributeValue DataType="string"
80                        >read</AttributeValue>
81       </Match>
82     </AllOf>
83    </AnyOf>
84 </target>
```

Listing 1: XACML 3.0 target example

For both versions of XACML, the AC logic described in listing 1 is represented exactly the same way in our non-technical notation as shown in Figure 2.

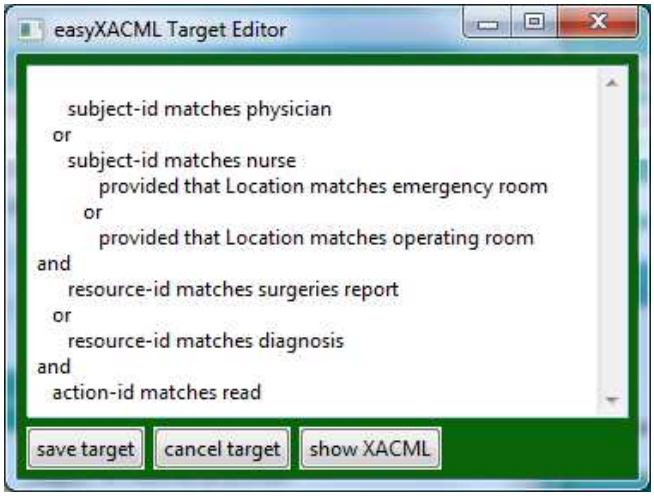


Figure 2: Non-technical notation rendering

III. Current Approaches to XACML Policy Administration

A steady evolution of approaches for editing XACML policies has taken place, starting with the most immediately obvious solution of using off-the shelf XML editors. However, the difficulty of the language associated with XML editors has led to three different categories of approaches:

- Use of dedicated XACML user interfaces

- Use of AC application custom made user interfaces
- Generic non-technical plain text XML-less editors

These three different approaches address two basic concerns:

- Reducing the learning curve of XACML through the use of pull down menus showing all XACML grammatical elements to choose from
- Making XACML totally transparent to the user

The most immediate solution for editing XACML policies is to use a generic XML editor with the XACML schema such as for example as the open source XmlPad [13]. In a case of a complex grammar like for XACML, the user loses the overview of what is required to be filled in, and more importantly, the overview of the logic expression he/she is attempting to assemble. XACML dedicated editors such as the University of Murcia open source UMU XACML editor [14] are essentially hard-coded XML graphical user interfaces.

We have also found that a significant number of industrial applications try to find a solution to the natural complexity of XACML by restricting the use of the rich capabilities of XACML. They do so by providing only a subset of XACML capabilities using simple logic structural concepts that satisfy the immediate needs of their users for specific AC applications. However, this approach will naturally result in non-re-usable applications because such GUIs need to be developed separately for each application. This also means that each of these separate applications needs to be maintained separately with the potential for duplication of human resources. Also, modifications to allow new AC needs will require long and costly design processes, thus limiting the possibility to adapt to new situations—particularly emergency situations—rapidly.

Another approach to avoiding working with XML is to use plain text languages. In [15], the authors propose an alternate syntax for the XACML language with the corresponding translators to pure XACML. A Technica project called noXACML [16] proposes a Java-like language where policies are described using plain if statements. Finally, other tools make use of the advantages of the Integrated Development Environment of Eclipse [17].

All of the above solutions have one thing in common. They require technical knowledge:

- Programming skills in general
- Technical knowledge of XACML structural elements

Thus, none of these solutions are usable for non-technical users.

IV. NON-TECHNICAL XML-LESS SOLUTIONS

By non-technical users, we mean those that understand only natural languages and have no experience with technical languages such as policy specification languages. Most access control policies are composed by such non-technical users, mostly at the managerial level and usually using natural languages. They are passed on to technical personnel that implement the policies in XACML. However, this process is often a source of errors due to potential misinterpretations. It also creates a technological barrier between the owner of a policy and the implementer, preventing the owner from verifying the correctness of the implemented policy. The need for non-technical approaches to policy making has been the subject of a survey in [18]. Although there have been experiments with translating policies written in natural language into formal languages [19], the natural ambiguity of natural languages reduces the effective usability of such an approach. New applications in the medical, military and emergency services domains require the capability to create new policies literally on the fly. Including technical people in this process is impossible because it causes too much delay in both the creation of policies and their verification. Instead, our approach consists of displaying AC logic in what appears to be natural language but in fact is very formal in the background and is constrained exclusively to the structural elements of XACML. This is easily achieved by coupling plain text with an object oriented (OO) representation of XACML policy elements using the XACML policy model provided by the standard, which is shown in Figure 3. Thus, in this XML-less plain text representation, there is a one-to-one mapping with XACML elements at all times. More important is the fact that the policy logic can only be displayed in plain text by deriving it from the background object oriented representation, and can actually never be written directly with a plain text editor. If the latter were allowed, this would mean that we were defining a new language and requiring the user to obtain the technical knowledge necessary to use it. This is exactly what we are trying to avoid. The internal object oriented representation is used to generate actual XACML code that is stored in a plain text file or a database. We note that this is the opposite of an approach presented in [20] where the natural language is transformed into a parse tree that is then manipulated to obtain a formal XQuery to an XML database.

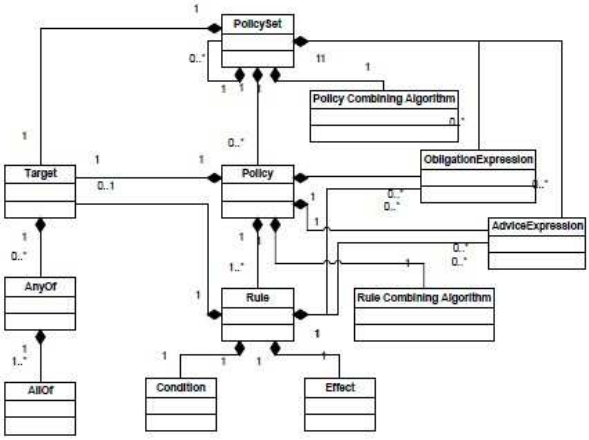


Figure 3: The XACML 3.0 policy model

A. *Principles of Interactive Plain Text*

The AC logic is rendered from an underlying object oriented representation. In order to be able to modify policy logic from that plain text representation, there needs to be a way to link it to the OO representation. This is easily achieved by recording the position of a plain text word in the text of the policy logic and linking it to its corresponding object instance. For example a rule target that states that a nurse can read a diagnosis report is rendered in our non-technical notation as follows:

```
   subject-id matches nurse
and
   resource-id matches diagnosis
and
   action matches read
```

It corresponds to an internal representation of a XACML 2.0 target as a hierarchy of object instances as shown in Figure 4.

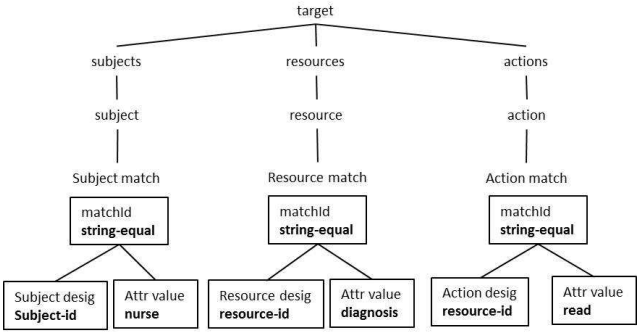


Figure 4: internal XACML object representation

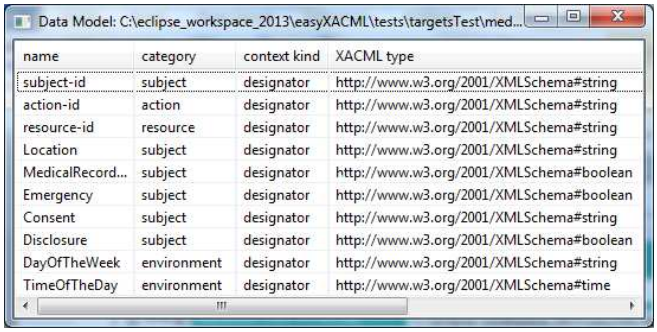
In our AC logic rendering, scoping of operators is achieved via indentation. This is a natural way of scoping, commonly used currently by non-technical users when writing plain text legal documents. In such documents a sub-clause is written using indented paragraphs usually preceded by clause numbers. Moreover, when constructing AC logic using our editor, non-technical users indicate scope by merely selecting an attribute value, and the placement of this value defines the context. For example, specifying that physicians or nurses can read surgery reports is rendered as follows:

```
      Subject matches physician
   or
      Subject matches nurse
and
   Resource matches surgery report
and
   Action matches read
```

In the above rendering, the conjunction operators are the top-level of the target expression while the disjunctions about the subjects represent the *AnyOf* target construct at the second level. Matches are specified as a simple sentence (not parse-tree like) because the scope of the match operator is clear enough in natural language.

B. *Principles of Data Models*

Hiding the complexity of XACML requires eliminating not only long XML tags but also long domain definitions for attribute values and even data values. This is easily achieved using data models as shown in Figure 5, which for each AC application attribute, describe the mapping between the displayed information and the XACML equivalent. Data models are widely used by all kinds of editors though usually only for determining the XML content of pull-down menus from which the user can select, such as an appropriate operator for a given type. Our data model contains additional information for the translation back and forth between the non-technical notation and real XACML. It also provides the capability for users to define their own operators and to redefine the external non-technical textual representation of any existing operator to their liking.



name	category	context kind	XACML type
subject-id	subject	designator	http://www.w3.org/2001/XMLSchema#string
action-id	action	designator	http://www.w3.org/2001/XMLSchema#string
resource-id	resource	designator	http://www.w3.org/2001/XMLSchema#string
Location	subject	designator	http://www.w3.org/2001/XMLSchema#string
MedicalRecord...	subject	designator	http://www.w3.org/2001/XMLSchema#boolean
Emergency	subject	designator	http://www.w3.org/2001/XMLSchema#boolean
Consent	subject	designator	http://www.w3.org/2001/XMLSchema#string
Disclosure	subject	designator	http://www.w3.org/2001/XMLSchema#boolean
DayOfTheWeek	environment	designator	http://www.w3.org/2001/XMLSchema#string
TimeOfTheDay	environment	designator	http://www.w3.org/2001/XMLSchema#time

Figure 5: XACML attribute data model

C. *Low Maintenance Cost*

One of the advantages of a generic XACML editor whether technical or non-technical that is not tied to a specific AC application is that it can be re-used for a wide number of applications across multiple companies without re-development. This provides the benefit of economies of scale both at the development and maintenance level. The only modification required is to the AC application's attribute data model. Modifying the data model is precisely the role of a technical person; in other words, a technical person must configure our editor so that it can be used by non-technical personnel. This principle applies both for technical and non-technical solutions.

V. NON-TECHNICAL TARGET EDITING

As discussed earlier, XACML targets require some duplication. In this section, we illustrate how we avoid such duplication when policies are presented to the user using our notation. In addition, as a user edits a policy, the need for duplication in the underlying XACML may come and go, but all such duplication, as it is added and removed, remains transparent to the user.

A. *Definition of Context*

Since each word in the non-technical expression rendering is linked to an OO representation, context can easily be derived by the underlying software. In our implementation,

context is defined by selecting the value of a match expression. Internally, the software will walk the object hierarchy and determine that this value is involved for example in a resource match. Once the context is determined, the software will offer the appropriate operation that can be performed at this point. For both XACML versions of targets, this consists of two possibilities:

- Adding an alternate group of matches (*AnyOf*)
- Refine an existing match with additional matches (*AllOf*)

For version 2.0, context also means determining the exact nature of the category of the attribute, i.e. subject, resource or action. Thus, in order to determine context, the user must first select a value and upon a right mouse click be presented with a menu of possible operations as shown in Figure 6. For example the fact that the value *nurse* was selected gives only the choice to add additional *AnyOf* constructs (*add an alternate constraint* menu item) or refine a category by adding additional *AllOf* constructs (*refine a category* menu item) but not *add a new constraint* in the highest level of the target parse tree.

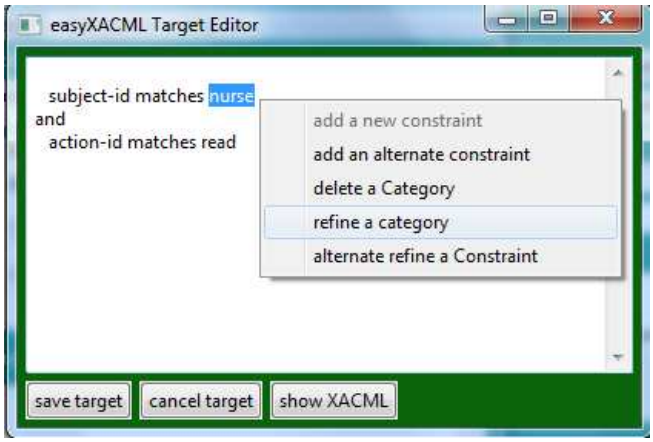


Figure 6: Determining context

Finally, when the user has selected a task that leads to the creation of a new expression, the context also determines the list of attributes and their related operators and potentially pre-defined values to construct a given match using the constraint editor as shown in Figure 7.

B. *Specifying a Match*

Matches are specified using the constraint editor interface that proposes a list of attribute names to choose from. This list is derived from the attribute data model specified by the policy administrator as shown in Figure 5. It is composed of three columns. The first column contains the list of attributes available from the attribute data model. Initially, the two remaining columns remain empty until the user selects one of the attributes. Once the user has selected an attribute to build her expression, a list of operators corresponding to the data type of the selected attribute will appear in column 2 and a list of predefined values that were defined in the data model will appear in column 3. This is an alternative approach to entering values directly. The concept of predefined values was introduced in order to further prevent errors due to spelling mistakes or domain violations. For example, in Figure 7, a user has chosen attribute resource-id that was defined in the data model as having the data type string, which then determines the list of available operators for that data type along with available predefined values.

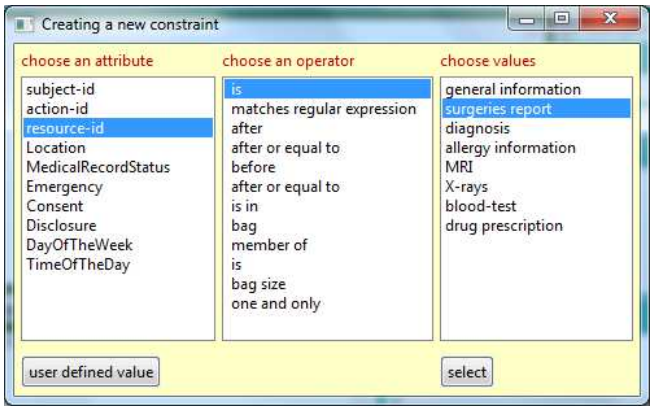


Figure 7: Non-technical constraint editor

Selecting attribute *resource-id*, defined as type string in the data model, and then selecting the corresponding match operator rendered with the word *matches*, and then the value *surgery report*, which we have pre-defined in the data model, will produce the corresponding objects in the internal representation and be rendered as follows:

```
resource-id matches surgery report
```

An action or subject can be selected using exactly the same procedure.

C. *Adding an Alternate Group of Matches*

Alternative groups of matches are represented in lists of categories (subject, resource or action) in version 2.0 and similar lists wrapped inside an *AnyOf* tag in version 3.0. In Figure 8, in order to add an alternate subject for physicians, the user first selects the current value *nurse* for attribute *subject-id* and upon a right mouse click, obtains the constraint editor where she is presented with the list of possible operations. To add an alternate constraint for physicians, the *add an alternate constraint* menu item would be selected to enter a new match for physicians. Once this operation completed, this will result in generating a new object instance in the underlying *AnyOf* construct as shown in listing 1 between lines 02 and 21. Both the nurse and the physician are wrapped around their own *AllOf* tags.

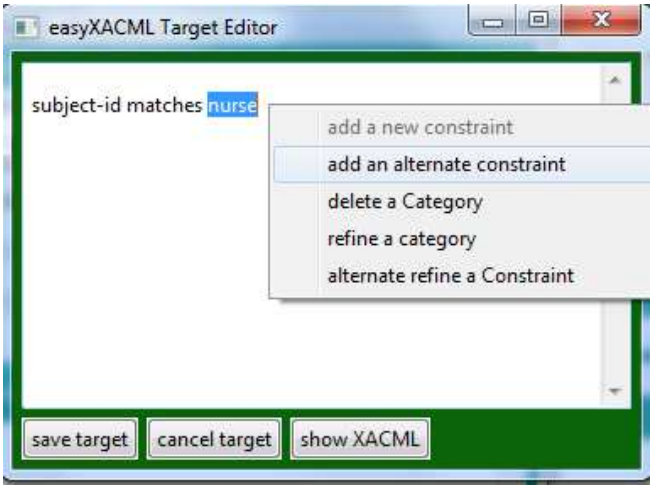


Figure 8: XACML 3.0, Adding alternate constraints

D. Refining a Match

Refining a match consists of adding additional matches such as for example stating that a nurse can read a diagnosis report only if he/she is located in an operating room. In version 2.0, this results in adding additional matches (subject, resource or actions) while in version 3.0 it consists of adding more matches inside an *AllOf* tag. The non-technical user does not need to be aware of these technical details and after selecting the value *nurse* and doing a right mouse click, merely selects the *refine a subject* menu item as shown in Figure 9. He/she will then be directed to the constraint editor to specify a match by selecting an attribute name, an operation and a value, for example the *Location* attribute and value *operating room*.

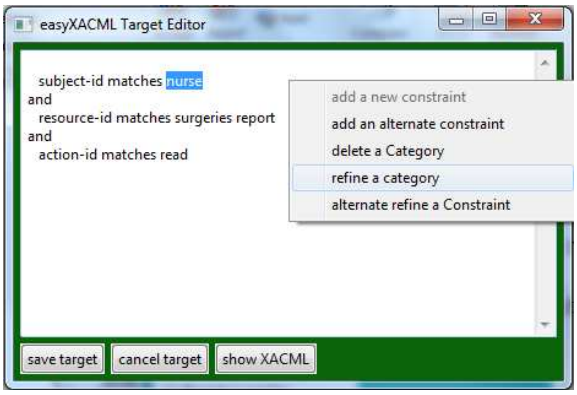


Figure 9: Refining a constraint

E. Hiding Implementation Details for Complex Expressions

The XACML target restrictions shown in Figure 1 force the user to use duplicate definitions when trying to construct more complex logic. For example, in listing 1, the logic that says that nurses can read diagnosis reports only while in an emergency room or operating room cannot be implemented as a disjunction at the lowest level of a target definition. This is because the disjunction provided by the XACML *AnyOf* language construct is already used up to distinguish the cases where the *subject-id* is either a physician or a nurse as shown at lines 20 and 38. The level below can contain only conjunctions as shown in Figure 1. Thus, in a XACML target, the work around this restriction can only be expressed by entering the *nurse* value for *subject-id* twice as two different alternatives and then using an additional subject match to distinguish the emergency room and operating room cases, as shown in Listing 1. The first conjunction between subject matches is found between lines 13 and 30 for the emergency room case and the second between lines 31 and 48 for the operating room case. The duplication of the *subject-id nurse* occurs at lines 20 and 38. We have determined that this restriction can be hidden to the non-technical user by factoring out the *nurse subject-id* value and creating a rendered disjunction at the lower level as shown in Figure 2. The generated XACML code is of course different than the rendering in this case but fully behaviorally equivalent. From a graphical interface point of view this is achieved with a special menu item called *alternate refine a category* as shown in Figure 10. When choosing this item, the software automatically inserts the duplicate intermediary level match (*nurse* in this example) but hides it in the rendered version as shown in Figure 2.

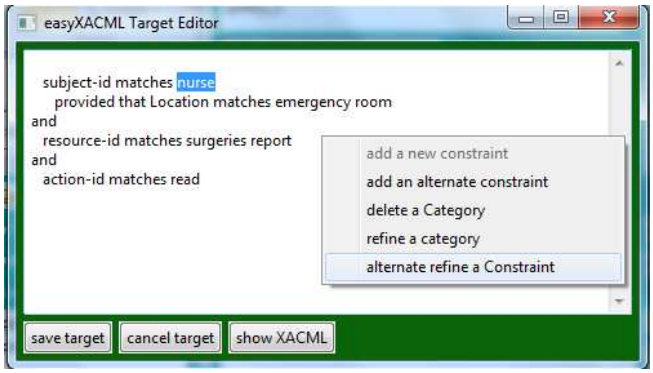


Figure 10: XACML 2.0, Entering an alternate refinement

Note that in this example, we have chosen a specification style where the logic is contained within a single policy. The most frequent style consists of specifying the two cases (*physician* and *nurse*) in two separate policies. This second approach actually does not avoid the duplication of the *subject-id nurse* either. The advantage of our first approach is that the entire logic is displayed at once in a single window and thus provides an overview of the logic.

A similar problem arises when deleting a match that is a refinement. When there are several matches in a subject, resource or action, there is no problem. We merely delete the selected match only. However, when there is only one additional match left, as for example to express that a nurse can read a surgery report in the case of an emergency, there are two different contexts to consider. The first one is the case where there is no alternative match starting with a duplication. This case can again be resolved by merely deleting the refinement. However, when there is a duplication of the first match among subjects, then deleting one of the refinement

matches would create a logical error because the second alternate match with the duplicate subject would make the first refined subject (the one containing more than one match) useless. For example, if we delete the *operating room* match in line 46 of Listing 1, the subject match *nurse* at line 38 would always pre-empt the first subject match at line 20 despite the subject match refinement *emergency room* at line 28 when evaluated by a PDP. Thus, in order to avoid such errors, and also to avoid the user having to remove both subject matches of the second subject in two steps, the editor merely removes the entire subject specification at once. Here again, we made a XACML technical detail completely transparent to the user in order to avoid errors.

The internal object oriented representation of this target is volatile. It exists only while the policy is loaded and displayed. Eventually it will be transformed from the internal object instance into XACML format upon saving.

VI. CONCLUSION

We have shown that it is easy to represent XACML targets using a non-technical notation and thus make the grammatical elements of XACML targets completely transparent to the non-technical user while editing a XACML policy. But we have also shown that implementation details to represent specific user requirements can also be made transparent to the user. While our new PAP implementation contributes to make XACML more usable in general, it also makes the migration from version 2.0 to the more powerful version 3.0 considerably easier. As stated, the new version is considered by many users as too complex, and thus our approach can eliminate the traditional fears that surround complexity. However, our non-technical approach to XACML policy editing still retains a technical element: the concept of policy set, policy, rule, targets, and conditions. So far we consider that these structural elements are easy to learn by a non-technical user, because they correspond to a very common and well understood structuring mechanism. However, the next step in this non-technical approach would be to make the above structural concepts transparent to the user as well.

ACKNOWLEDGMENT

Our thanks to numerous industrial partners that we cannot name due to non-disclosure agreements.

REFERENCES

[1] XACML 2.0 OASIS standard DOI= http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[2] XACML 3.0 OASIS standard DOI= http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf

[3] V.C. Hu, D. Ferraiolo, R. Kuhn, n, A. Schnitzer, K. Sandlin, R. Miller, R. Miller, K. Scarfone, Guide to Attribute Based Access Control (ABAC) Definition and Considerations, in NIST Special Publication 800-162

[4] S. Verma, M. Singh, S. Kumar, Comparative analusis of Role Base and Attribute Base Access Control Model in Semantic Web, in International Journal of Computer Applications, vol. 46, No. 18, May 2012.

[5] D. F. Ferraiolo, D. R. Kuhn, Role-based access control, in *Proc. of the 15th National Computer Security Conference*, pages 554-563,1992.

[6] G.Leighton, D. Barbosa, Access Control Policy Translation and Verification within Heterogeneous Data Federations, in Proceedings of the 15th ACM symposium on Access control models and technologies, pages 173-182.

[7] W.V. Sujansky, S. A. Faus, E. Stone, P. Flatley Brennan, A method to implement fine-grained access control for personal health records through standard database queries in Journal of Biomedical Informatics, pages S46-S50.

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

[9] B. Stepien, A. Felty, S. Matwin, Advantages of a Non-Technical XACML Notation in Role-Based Models, in PST 2011 proceedings

[10] A. Anderson, S. Proctor, Method for analysing an XACML policy, U.S. patent 20100042973.

[11] S. Jahid, C.A. Gunter, I. Hoque, H. Okhravi, MyABDAC: Compiling XACML Policies for Attribute-Based Database Access Control in CODASPY'11 proceedings, pages 97-108.

[12] G. Karjoth, A. Schade, Implementing ACL-Based Policies in XACML, in ACSAC'08 proceedings, pages 183-192.

[13] XmlPad, http://xml-pad.sourceforge.com

[14] UMU editor, http://sourceforge/net/projects/umu-xacmleditor

[15] M. Masi, R.Pugliese, F. Tiezzi, Formalisation and Implementation of the XACML Access Control Mechanism, in proccedings of ESSoS'12 of the 4th international conference on Engineering Secure Software and Systems, pages 60-74.

[16] Technica, NOXACML project, http://bradjcox.blogspot.ca/2012/06/i-just-got-ok-to-start-open-source.html

[17] Axiomatics, Axiomatics Language for Authorization, DOI: http://www.axiomatics.com/axiomatics-alfa-plugin-for-eclipse.html

[18] L. Bauer, L. Faith Cranor, R.W. Reeder, M. K. Reiter, and K. Vanica, "Real life challenges in access-control management," 27th CHI Conference, 2009.

[19] C.A. Brodie, C.-M. Karat and J. Karat, "An empirical study of natural language parsing of privacy policy rules using the Sparcle policy workbench," Proc. of the 2nd Symposium on Usable Privacy and Security, 2006.

[20] Y. Li, H. Yang, H. Jagadish, Constructing a generic natural language interface for an XML database, in International Conference on Extending Database Technology proceedings.