# On the Use of OWL Reasoning for Evaluating Access Control Policies

Fabio Marfia
Politecnico di Milano
DEIB - Department of Information, Electronics and Bioengineering
Via Ponzio, 34/5
20133, Milano - Italy
fabio.marfia@polimi.it

Mario Arrigoni Neri
University of Bergamo
Department of Computer Engineering and Mathematical Methods
viale Marconi, 5
24044, Dalmine (BG) - Italy
mario.arrigonineri@unibg.it

Filippo Pellegrini
Politecnico di Milano
DEIB - Department of Information, Electronics and Bioengineering
Via Ponzio, 34/5
20133, Milano - Italy
filippo1.pellegrini@mail.polimi.it

Marco Colombetti
Politecnico di Milano
DEIB - Department of Information, Electronics and Bioengineering
Via Ponzio, 34/5
20133, Milano - Italy
marco.colombetti@polimi.it

## ABSTRACT

We present a Description Logics approach to the management of XACML policies. We explain how policies can be mapped to a DL axiomatization, and how authorization requests can be answered using standard DL reasoning tools. Our model represents a valid substratum for managing policies whose expressivity can not be captured by standard engines. Furthermore, advanced security functionalities, as Policy Harmonization and Policy Explanation, can be implemented in the context of the present model.

## Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access Controls

## General Terms

Management, Security

## Keywords

Access Control, Policy Languages, Description Logics, Reasoning

## 1. INTRODUCTION

As data engineers decided to create application-independent logical mechanisms of data storage in the '60s, and first database systems became available, the chance of modeling logically-independent policy management systems has been considered in the last decade.

As a matter of facts, an increasing number of distributed applications have been meeting complex problems in managing distribution and access authorizations of contents in the last years. Description and enforcement of policies can become a very complex task in large systems. An independent Policy Management architecture represents a scalable and re-usable environment for: formally specifying policies (Policy Editing and Storing); automatically asserting, according to the specified policies, whether an agent is authorized to commit an act, or not (Policy Evaluation); automatically resolving conflicts between policies (Policy Harmonization); generating human-readable explanations of the causes of a specific policy evaluation (Policy Explanation); eventually enforce an agent to commit an act or perpetrate penalization acts against agents, as a response to policy violations (Policy Enforcement).

Several standard languages have been defined for the formal representation of policies in such type of environment. However, in fact, the current *de facto* standard in access control policy languages is represented by XACML [8].

XACML defines standard protocols for transmitting credentials, requesting resources, defining and storing access policies; together with the definition of a general security layer, made up of different and specialised software components [7]. Such a layer deals with the aforementioned tasks of allowing policy administrators to edit and store policies, handling conflicts between contradictory decrees, providing a ultimate response for access requests, together with an explanation of such a response eventually.

We present an implementation of an XACML-compliant framework in this paper, based on OWL and reasoning technologies. The expression and application of deontic propositions is well known in literature, however, as far as we know, this is the first time they are applied with the specific aim of providing a solution for an XACML security layer, even if activities for formalizing XACML policies with Description Logics (DL) were done in the past, for Policy Harmonization purposes [3].

Relying many core functionalities on DL reasoning activities, performances result worse than any known XACML engine's. Consequently, the present approach has to be considered in cases where current XACML technologies are unable to capture the expressiveness of the policies involved, or as a substratum for providing advanced functionalities as, e.g., Policy Explanation.

The paper proceeds as follows: we present the modular structure of the framework in Section 2. We explain how DL axioms are generated from a collection of XACML policies, and how policy evaluation is done in Section 3.

## 2. THE POLICY FRAMEWORK

As described in [7], the XACML standard defines a general framework for receiving data requests and handling responses according to an arbitrarily large collection of policies, that are stored in a repository according to a standard XML-compliant model. The subsequent architectural components are defined for the framework:

- **Policy Enforcement Point (PEP)**: Point which intercepts user's access request to a resource, makes a evaluation request to the PDP to obtain the access evaluation (i.e. access to the resource is approved or rejected), and acts on the received evaluation;
- **Policy Decision Point (PDP)**: Point which evaluates access requests against authorization policies before issuing access responses;
- **Policy Administration Point (PAP)**: Point which manages access authorization policies;
- **Policy Information Point (PIP)**: The system entity that acts as a source of attribute values;
- **Context Handler**: the Context Handler deals with the coordination of the communications between PDP, PEP and PIP; in particular, it acts in order to return the output of the PDP to the PEP as a response for an access request, consisting eventually in a retrieved resource.

The technology behind the PEP, that is developed in order to enforce or regulate access to resources, is strongly domain dependent and it is not matter of the current work. The PDP is provided with a Policy Evaluator component that interfaces with a DL Reasoner. Policy evaluations and explanations are generated by the PDP as a result of a reasoning activity on three different ontologies:

1. A **Policy Terminological Box** (TBOX), that is the expression of the active policies and it is obtained as a result of an algorithmic translation from an XACML collection of policies. Tasks such as providing an interface for policy editing to policy administrators, synchronizing the TBOX ontology with XACML policies, harmonizing conflict between policies, are delegated to the PAP.
2. A **Domain TBOX**, representing a meaningful portrayal of the application domain. It is arbitrarily expressive and it is thought to cover the whole collection of concepts and relations involved in the application domain.
3. A **Domain Assertional BOX** (ABOX), gathering the different descriptions of the individuals and resources involved in the application domain. They are represented as an instantiations of the concepts and relations depicted in the Domain TBOX.

Both Domain TBOX and ABOX are stored and managed by the PIP.

## 3. POLICIES AS AXIOMATIC PROPOSITIONS

As presented in Section 2, the PAP allows a policy administrator to edit and store policies in the form of an XACML collection. XACML represents the XML-compliant description of the policies in the environment, while the DL form of the same collection is represented by an OWL TBOX. Policies are translated from the former representation to the latter automatically.

Kolovski et al. [3] present how to formalize XACML policies, using a more complex syntax than DL, defined DDL$^-$. That is done according to three types of XACML combining algorithms (see also [7]): *permit-overrides*, *deny-overrides*, *first-applicable*. We decided to reduce the expressivity of the XACML collection specifiable by the PAP, in respect to the aforementioned formalisation, as follows: the policy collection is reduced to a set of XACML rules, applying according to the policy combining algorithm *deny-overrides* only. The algorithm takes into consideration every rule, and, then, if both access deny and permit apply, an access deny is returned as a response. Whether nothing is found to be applied in the whole set of rules, a final general policy is defined in order to deny any access. Such approach allows to rely on standard DL technologies for reasoning without involving the DDL$^-$ formalisation, obtaining better performances and an easier policy representation. We believe that such simplification is a sufficient approach for satisfying the requirements of many real-life environments in which, for security reasons, every access is denied *ex-ante*, while policies are applied for modifying such default behaviour.

In order for the rules to be properly translated into an OWL TBOX, they can not be expressed arbitrarily: we have then identified five different policy archetypes, according to which the policies must be defined. The identified archetypes cover a wide range of expressiveness, in particular the three standard models IBAC (Identity Based Access Control), RBAC (Role Based Access Control) and ABAC (Attribute Based Access Control) are covered by the model.

The five different policy archetypes are shown in Table 1. Each policy can be composed with others using AND or OR conjunctions in our model, as foreseen by XACML protocol, in order to generate complex rules. Furthermore, each

Table 1: Policy Archetypes

| ID | Access Control Reference | Description | Example |
|---|---|---|---|
| 1 | IBAC | A single subject is allowed to access to one or more resources | John Andrews can read Healthcare Assistant Documents |
| 2 | RBAC | A group of subjects is allowed to access to one or more resources | Medical Consultants can write a Medical Regulation Document |
| 3 | ABAC | Only subjects with specific attributes are allowed to access to one or more resources | Females can not read Andrology Documents |
| 4 | ABAC | Only subjects in a specific relation with another subject with specific attributes are allowed to access to one or more resources | A tutor of a person that is not of age can read document 305871 |
| 5 | N/A | Only subjects in a specific relation with another subject are allowed to access to the resources that refer to the latter subject | A subject can read all the records of the ward he/she works in |

policy can be positive or negative, allowing the policy administrator to permit or deny access to specific resources.

We describe two between the five policy archetypes, together with policy examples, in Section 3.1, while specifying how each type of policy is translated into a set of axioms. We describe how policies can be combined with AND or OR conjunctions in Section 3.2. We present how the PDP can obtain policy evaluations from the generated axioms in Section 3.3.

## 3.1 Policy Archetypes

### *3.1.1 Type 3 - ABAC Simple Policy*

The Type 3 archetype represents the permission released or denied to a single subject characterized by one or more attributes, for the access to a resource or a group of resources. A sample XACML Type 3 rule is shown in Table 1, allowing every subject with dataProperty `hasGender` equal to `"F"` to `read` the group of resources `AndrologyDocument`. The policy is translated into a TBOX ontological policy with the subsequent procedure. First, an OWL class is generated, containing only the individuals characterized by the aforementioned property:

```
Class:
   hasGender_F_Class
equivalentTo:
   hasGender value "F"
```

Then, the functional Identity Property `identityOn_hasGender_F_Class` is defined for the generated class, representing the property of each member of the class pointing to the member itself:

```
Class:
   hasGender_F_Class
equivalentTo:
   identityOn_hasGender_F_Class some Self
```

The same is done for the group of resources, represented in the Domain TBOX ontology by the class `AndrologyDocument`:

```
Class:
   AndrologyDocument
equivalentTo:
   identityOn_AndrologyDocument some Self
```

Finally, the negative permission to be annotated, `CanNotRead`, is defined as a superproperty of a specific property chain, as follows:

```
objectProperty:
   identityOn_hasGender_F_Class o
   topObjectProperty o
   identityOn_AndrologyDocument
SubPropertyOf:
   CanNotRead
```

All the individuals with the dataProperty `hasGender value "F"` are connected in this way with the property `CanNotRead` to each resource belonging to the class `AndrologyDocument`.

### *3.1.2 Type 5 - Triangular Relation Policy*

The Type 5 policy archetype puts in relation the subject and the resource generating the permission only in the case that a common individual is in a specific connection with both of them. The added permission property represents the side of a triangle in such a case, where its vertexes are represented by the subject, the resource and the individual in common. A sample Type 5 rule is shown in Table 1, allowing every subject to `read` every `MedicalRecord` of the `ward` in which he `worksIn`.

The OWL axiomatic expression of such a policy is characterized by a single property chain expressed as a subproperty of the involved permission property:

```
objectProperty:
   worksIn o
   ownsRecord
SubPropertyOf:
   canRead
```

## 3.2 Combining Policies

As stated, the presented policies can be also expressed together in a single XACML rule using AND or OR operators. As it can be understood, each archetype differs from the others for the way in which the subject requirements are expressed only, while the expression of permission and resources (a single one, or a group) are the same. So, a joined expression of two policies can be, for example, allowing a **Medic** that is **male** to read every `AndrologyDocument`. That is an expression of a Type 2 + Type 3 policy.

In case that many policies are joined with an OR conjunction, it is sufficient to translate each policy singularly into a TBOX policy. In case that many policies are joined with an AND conjunction, the approach changes whether none, one or more Type 5 policy are present. In case that no Type 5 policy is present, a new class is defined as an intersection of all the classes that identify the subject requirements for every policy. Then, a new Identity Property is created for the class and the positive or negative permission is assigned as a superproperty of the property chain between the created Identity Property, the `topOjbectProperty` and the Identity Property on the resources, as it is done for any of the Type 1 to 4 archetypes.

In case that one Type 5 policy is present, a new class is defined as an intersection of all the classes that identify the subject requirements for every policy of Type 1 to 4. Then, a new Identity Property is created for the class and the positive or negative permission is assigned as a superproperty of the property chain between the just created Identity Property and the two object properties involved in the Type 5 policy.

Expression of an axiomatic policy is not possible, using DL, in case of AND conjunction between more than one Type 5 policies. That because specification of double paths between the same identities is involved, and it is not possible. However, the issue can be addressed using SWRL Rules [6].

### 3.3 Policy Evaluation

Once the XACML policies are correctly translated into a Policy TBOX by the PAP, when the PDP receives an XACML access request from the Context Handler (more formally, an XACML *Context* [7]), it retrieves the Policy TBOX from the PAP and the Domain TBOX and ABOX from the PIP.

After that, the task of Policy Evaluation reduces itself to the process of querying the set of the retrieved ontologies, for verifying whether they logically entail or not two specific theorems: the one stating that the subject *can do* the requested action on the requested resource (*positive permission theorem*), and the one stating that the subject *can not do* the requested action on the requested resource (*negative permission theorem*).

Whether only the positive permission theorem is found, a positive authorization is returned by the PDP to the Context Handler. A negative authorization is returned in any other case. As an example, we can assume that the permission request for `john_andrews` to `read` the document `document_305871` is received by the PDP from the Context Handler. Two DL queries [1] are sent, then, to the set of ontologies for retrieving the two subsequent theorems:

1. `john_andrews canRead document_305871.`
2. `john_andrews canNotRead document_305871.`

If theorem 1 is found only, the response of the PDP is a positive authorization. Otherwise, the response is a negative authorization.

## 4. CONCLUSION AND FUTURE WORK

We measured the algorithm performances in respect to the best known XACML engines [5]. While policy conversion to DL is executed within a reasonable time, Policy Decision performances are worse than any known XACML engine, in the order of 100x in time approximately. Anyway, we believe that our solution can be a reasonable alternative in a real-life scenario to ordinary XACML engines for the subsequent reasons:

- An optimized code and an efficient hardware can support a usable PDP engine for real-time interactions in real-life environments;
- DL expressiveness can be used to define policies which complexity can not be caught by ordinary XACML engines;
- External applications can generate interesting portrays of the regulation state by accessing to the framework semantics for an end-user;
- Automatic agents may regulate their behaviour by reading and reasoning on provided policies;
- Advanced complex tasks can be exploited that are almost impossible for ordinary XACML frameworks; as Policy Explanation, or Policy Harmonization.

Policy Explanation can be provided together with the response using OWL Explanation technology [2], as already done by Marfia [4]. A Policy Harmonization service based on the OWL policy representation can be developed for the framework, accordingly to what presented by Kolovski et al. [3].

## 5. REFERENCES

[1] DL Query guide - Protégé DLQueryTab. `http://protegewiki.stanford.edu/wiki/DLQueryTab`, 2008.

[2] M. Horridge, B. Parsia, and U. Sattler. Laconic and Precise Justifications in OWL. In *Proceedings of the 7th International Conference on The Semantic Web*, ISWC '08, pages 323–338, Berlin, Heidelberg, 2008. Springer-Verlag.

[3] V. Kolovski, J. Hendler, and B. Parsia. Analyzing Web Access Control Policies. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 677–686, New York, NY, USA, 2007. ACM.

[4] F. Marfia. Using Abductive and Inductive Inference to Generate Policy Explanations. In M. Obaidat, A. Holzinger, and P. Samarati, editors, *Proceedings of International Conference on Security and Cryptography (SECRYPT 2014)*. SciTePress, 2014.

[5] A. Mourad and H. Jebbaoui. SBA-XACML: Set-based approach providing efficient policy decision process for accessing Web services. *Expert Syst. Appl.*, 42(1):165–178, 2015.

[6] SWRL: A Semantic Web Rule Language Combining OWL and RuleML. `http://www.w3.org/Submission/SWRL/`, 2004.

[7] OASIS XACML Version 3.0 Specification. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf`, 2013.

[8] OASIS eXtensible Access Control Markup Language (XACML). `https://www.oasis-open.org/committees/xacml/`, 2013.