An OWL-based XACML Policy Framework

Fabio Marfia[1,2], Mario Arrigoni Neri[3], Filippo Pellegrini[1] and Marco Colombetti[1,2]
[1]*DEIB - Department of Information, Electronics and Bioengineering, Politecnico di Milano, Via Ponzio 34/5, Milano, Italy*
[2]*Faculty of Communication Sciences, Università della Svizzera italiana, Via G. Buffi 13, Lugano, Switzerland*
[3]*Department of Computer Engineering and Mathematical Methods, University of Bergamo, viale Marconi 5, Dalmine, Italy*

Keywords: Access Control, Policy Languages, Description Logics, Reasoning.

Abstract: We present an XACML policy framework implementation using OWL and reasoning technologies. Reasoning allows to easily generate policy decisions in complex environments for expressive policies, while satisfying the requirements of reliability and consistency for the framework. Furthermore, OWL ontologies represent a valid substratum for tackling advanced complex tasks, as Policy Harmonization and Explanation, with a complete rationale.

1 INTRODUCTION

Policies are a widespread approach for protecting users privacy and security, and for allowing or enforcing users to abide by different norms and laws. More and more complex and distributed scenarios are requesting access control solutions, in the last years, where a centralized framework is required to manage a large set of functionalities related to policies. Those functionalities include policy editing, storing, harmonization, decision making, explanation.

Different standards are presented in literature, modeling the general architectures and language protocols involved in such a type of framework. As presented by (Ardagna et al., 2011), while different solutions where developed during the last decade, the current *de facto* standard in access control policy languages is represented by XACML (XACML Standard, 2013).

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 (XACML Data-Flow model, 2013). 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, see, e.g., (Singh, 1998), (López et al., 2006), (Fornara and Colombetti, 2010). 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 (Kolovski et al., 2007).

The present approach allows the expression of complex and expressive policies as it is requested, we believe, from nowadays pluralistic scenarios. Furthermore, a knowledge base represents an interesting representation for policies, alternative to XACML, for advanced automatic components wishing to portray the regulation state for an end-user, or components that are intended to modify their behaviour by reasoning on provided directives. All of that, while allowing policy administrators to edit and store policies without revising any hard-coded software.

Relying many core functionalities on DL reasoning activities requiring significant computational resources, performances are worse than any known XACML engine's, as presented in Section 4. 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. More details on the subject are presented in Section 6.

Every policy example that is considered in this pa-

Marfia F., Arrigoni Neri M., Pellegrini F. and Colombetti M..
An OWL-based XACML Policy Framework.
DOI: 10.5220/0005570101240135
In *Proceedings of the 12th International Conference on Security and Cryptography* (SECRYPT-2015), pages 124-135
ISBN: 978-989-758-117-5

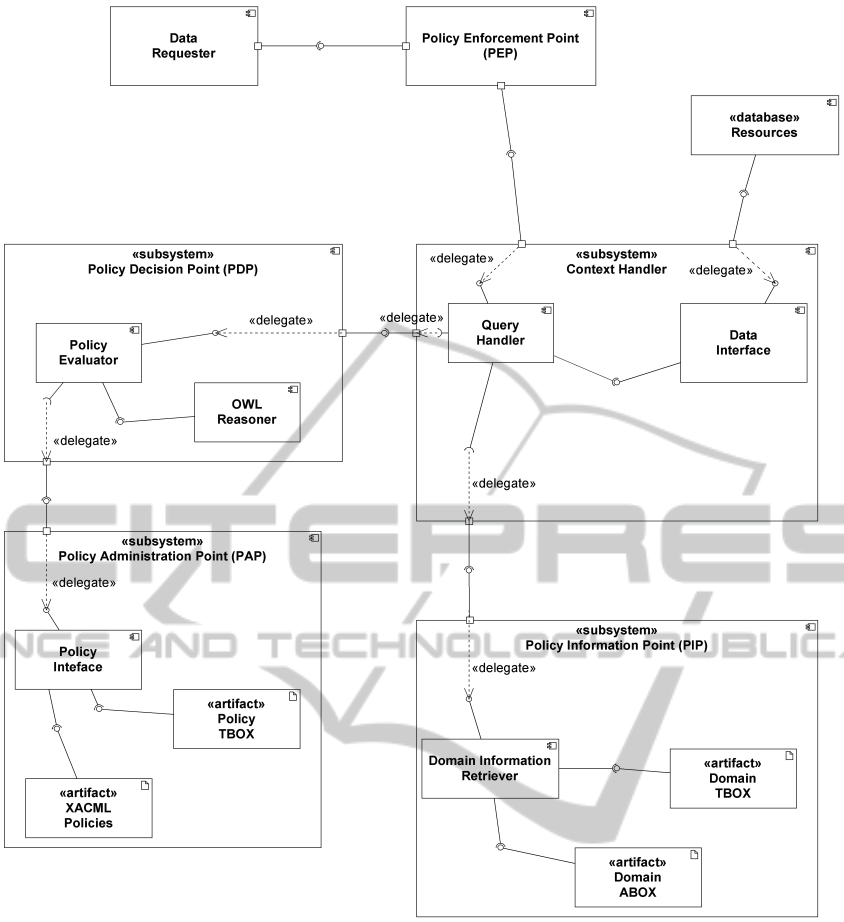

Figure 1: OWL-based XACML Framework – Component Diagram.

per is taken from hospital unit use cases, representing a real-life scenario in which the wide policy expressivity used in the present work definitely represents a necessary requirement for data security and privacy.

The paper proceeds as follows: we present the components and the structure of the framework in Section 2. We explain how XACML policies are converted into a collection of DL axioms, and how policy decisions are generated in Section 3. Performances are evaluated in Section 4. Related work is presented in Section 5. Conclusion and future work are presented in Section 6.

2 AN OWL-BASED XACML POLICY FRAMEWORK

As described in (XACML Data-Flow model, 2013), 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, as described in (XACML RFC2904, 2000):

- **Policy Enforcement Point (PEP)**: Point which intercepts user's access request to a resource, makes a decision request to the PDP to obtain the access decision (i.e. access to the resource is approved or rejected), and acts on the received decision;
- **Policy Decision Point (PDP)**: Point which evaluates access requests against authorization policies before issuing access decisions;
- **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.

Figure 1 presents how the described components interact with one another, while showing in a synoptic view each component's outline in the OWL-based implementation.

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 decisions 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 AXIOMATIC POLICIES

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., 2007) 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 (XACML Combining Algorithms, 2013)): *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 IBAC, RBAC and ABAC are fully 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 policy can be positive or negative, allowing the policy administrator to permit or deny access to specific resources.

We describe the five policy archetypes, together with policy examples, in Section 3.1, while specifying how every 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 the general algorithm for translating XACML policies in Section 3.3. We present how the PDP can obtain policy decisions from the generated axioms in Section 3.4.

The syntax in which all DL axioms are presented in this Section is the Manchester OWL Syntax (Horridge and Patel-schneider, 2008).

It is important to underline that complex DL structures (i.e. property chains, class definitions) can be present in the Domain TBOX, allowing the definition of policies based on composite relations in a concise manner. So, the reader must not be mislead in considering the whole expressive ability of the system

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

by the simple archetypes shown. In principle, every single proposition in a policy could be obtained after a complex inference procedure, starting from the axioms expressed in the Domain TBOX and ABOX.

For example, we may want to define the tutor of an impatient *I* as herself, if she is of age; otherwise, her father and mother are her tutor. In order to express a policy allowing the tutor of an impatient to access to certain documents, we can simply define the subject of the policy as `tutorOf` *I*. That can be done as DL axioms are present in the Domain TBOX defining what a tutor is, according to the aforementioned definition.

3.1 Policy Archetypes

3.1.1 Type 1 - IBAC Simple Policy

The first policy archetype is the permission released or denied to a single subject, for the access to a resource or a group of resources. A sample XACML Type 1 rule is shown in Table 1, allowing the subject `John Andrews` to `read` the group of resources `HealthcareAssistantDocument`. The policy is translated into a TBOX ontological policy with the subsequent procedure. First, an OWL class is generated, containing only the individual `john_andrews`:

```
Class:
   John_andrews_Class

equivalentTo:
   {john_andrews}
```

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

```
Class:
   John_andrews_Class

equivalentTo:
   identityOn_John_andrews_Class some Self
```

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

```
Class:
   HealthcareAssistantDocument

equivalentTo:
   identityOn_HealthcareAssistantDocument
some Self
```

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

```
objectProperty:
   identityOn_John_andrews_Class o
   topObjectProperty o
   identityOn_HealthcareAssistantDocument

SubPropertyOf:
   canRead
```

where `topObjectProperty` is the Universal Property, connecting each entity to each other in the ontology. The individual `john_andrews` is connected in this way with the property `canRead` to each resource belonging to the class

HealthcareAssistantDocument.

3.1.2 Type 2 - RBAC Policy

The second policy archetype is a permission released or denied to a group of subjects, for the access to a resource or a group of resources. A sample XACML Type 2 rule is shown in Listing 1, allowing every identity that is a MedicalAssistant to write a document that is a MedicalRegulationDocument. The policy is translated into a TBOX ontological policy with the subsequent procedure. First, the functional Identity Property identityOn_MedicalAssistant is defined, representing the property of each member of the class MedicalAssistant pointing to the member itself:

```
Class:
   MedicalAssistant

equivalentTo:
   identityOn_MedicalAssistant some Self
```

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

```
Class:
   MedicalRegulationDocument

equivalentTo:
   identityOn_MedicalRegulationDocument
   some Self
```

Finally, the positive permission to be annotated, canWrite, is defined as a superproperty of a specific property chain, as follows:

```
objectProperty:
   identityOn_MedicalAssistant o
   topObjectProperty o
   identityOn_MedicalRegulationDocument

SubPropertyOf:
   canWrite
```

All the individuals belonging to the class MedicalAssistant are connected in this way with the property canWrite to each resource belonging to the class MedicalRegulationDocument.

3.1.3 Type 3 - ABAC Simple Policy

The third policy 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 Listing 2, 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.

3.1.4 Type 4 - ABAC Composite Policy

The fourth policy archetype differs from the simple ABAC policy as the permission is released or denied to a subject that is in relation with another subject characterized by one or more attributes, for the access to a resource or a group of resources. So, attributes are checked for the latter subject, while the permission is released or denied to the former. A sample XACML Type 4 rule is shown in Table 1, allowing every subject that `isTutorOf` another subject with dataProperty `hasAge` lower than `18` to `read` the single document `document_305871`. 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 relation:

```
Class:
   isTutuorOf_subjectThat_hasAge_
   lowerThan18_Class
equivalentTo:
   isTutorOf some (hasAge some int[< 18])
```

Then, the functional Identity Property `identityOn_isTutuorOf_subjectThat_hasAge_lowerThan18_Class` is defined for the generated class, representing the property of each member of the class pointing to the member itself:

```
Class:
   isTutuorOf_subjectThat_hasAge_
   lowerThan18_Class

equivalentTo:
   isTutuorOf_subjectThat_hasAge_
   lowerThan18_Class some Self
```

Being the resource a single resource, an OWL class has to be generated for it:

```
Class:
   document_305871_Class

equivalentTo:
   {document_305871}
```

Then, the functional Identity Property `identityOn_document_305871_Class` is generated as well for the class:

```
Class:
   document_305871_Class
equivalentTo:
   identityOn_document_305871_Class
   some Self
```

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

```
objectProperty:
   isTutuorOf_subjectThat_hasAge_-
   lowerThan18_Class o
   topObjectProperty o
   identityOn_document_305871_Class

SubPropertyOf:
   CanRead
```

All the individuals with the objectProperty `isTutuorOf` pointing to an individual with the dataProperty `hasAge some int[< 18]` are connected in this way with the property `CanRead` to the document `document_305871`.

3.1.5 Type 5 - Triangular Relation Policy

The fifth 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 and in Figure 2, 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 ex-

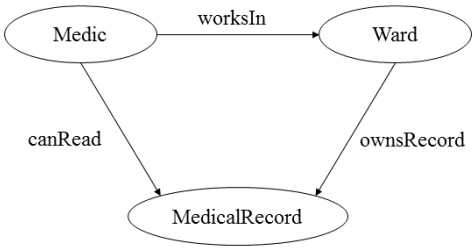

Figure 2: Type 5 policy example – the annotated permission is the side of a triangle in the ontological representation.

pression 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, as explained in (Hitzler et al., 2009). However, it is possible to address the issue defining a SWRL Rule (SWRL Rules, 2004).

As an example, we may want to allow a medic to `read` any medical record of the ward he `worksIn`, but only if it is about a patient he follows. That is an AND conjunction between two Type 5 policies. The subsequent SWRL rules could then be defined:

```
worsIn(x, z), ownsRecord(z, y),
isMedicOf(x, a), isAbout(y,a)
-> canRead(x, a)
```

3.3 Axiomatic Policies Generation Algorithm

We present the general algorithm for translating the XACML collection of policies into an OWL TBOX, as Algorithm 3.1.

Algorithm 3.1: XACML_POLICIES_TO_DL_POLICIES()

O = *loaded ontology (Policy TBOX)*
$\mathcal{R}$ = *set of rules*
for each $\langle rule : r \rangle \in \mathcal{R}$
do
- $\mathcal{TS} = \emptyset$ (*set of subject identities for rule r*)
- $\mathcal{S}$ = *set of subjects of rule r*
- **for each** $\langle subject : s \rangle \in \mathcal{S}$ **do**
  - $\mathcal{TM} = \emptyset$ (*set of subjectMatch identities for subject s*)
  - $\mathcal{M}$ = *set of subjectMatches of subject s*
  - **for each** $\langle subjectMatch : m \rangle \in \mathcal{M}$ **do**
    - *create objectProperty i as identity on m*
    - $\mathcal{TM} \leftarrow \mathcal{TM} \cup \{i\}$
  - *create objectProperty c as identity on the conjunction of all elements in* $\mathcal{TM}$
  - $\mathcal{TS} \leftarrow \mathcal{TS} \cup \{c\}$
- $\mathcal{TD} = \emptyset$ (*set of resource identities for rule r*)
- $\mathcal{D}$ = *set of resources of rule r*
- **for each** $\langle resource : d \rangle \in \mathcal{D}$ **do**
  - $\mathcal{TM} = \emptyset$ (*set of resourceMatch identities for resource d*)
  - $\mathcal{M}$ = *set of resourceMatches of resource d*
  - **for each** $\langle resourceMatch : m \rangle \in \mathcal{M}$ **do**
    - *create objectProperty i as identity on m*
    - $\mathcal{TM} \leftarrow \mathcal{TM} \cup \{i\}$
  - *create objectProperty c as identity on the conjunction of all elements in* $\mathcal{TM}$
  - $\mathcal{TD} \leftarrow \mathcal{TD} \cup \{c\}$
- a : *objectProperty for the action specified in rule r*
- U : *the Universal Property (topObjectProperty)*
- **for each** $\langle objectProperty : ts \rangle \in \mathcal{TS}$ **do**
  - **for each** $\langle objectProperty : td \rangle \in \mathcal{TD}$ **do**
    - *axiom* $ax = \{ts \circ U \circ td \sqsubseteq a\}$
    - $O \leftarrow O \cup \{ax\}$

The procedure analyzes rule by rule and, for each rule, the approach for managing it can be divided into three main parts.

First, each `subject` *s* in the rule is analyzed: each `subjectMatch` *m* in *s* is taken into consideration for defining an OWL class, that corresponds to *s*. The definition of such class is obtained as the conjunction of all the `subjectMatch`es in *s*.

Then, the same is done for each `resource` *r*: each `resourceMatch` in *r* is taken into consideration for defining a class, that corresponds to *r*. The definition of such class is obtained as the conjunction of all the `resourceMatch`es in *r*.

The final part considers each `(subject, resource)` couple, and it generates a policy axiom *ax* defining the property chain between the identity on the `subject` class, the universal property *U* and the identity on the `resource` class as a subclass of the action permission *a*, as specified in the rule (it may be, e.g., `canRead`).

3.4 Policy Decision

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* (XACML Data-Flow model, 2013)), it retrieves the Policy TBOX from the PAP and the Domain TBOX and ABOX from the PIP.

The Domain ABOX may include even thousands individuals in real environments, so, only individuals that can be useful for inferring the access permission are kept in the ABOX by the PIP before returning it. Then, the task of Policy Decision 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 (DL query, 2008) 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 PERFORMANCES

We developed a prototype framework using JAVA and OWL API (Horridge and Bechhofer, 2011), with Hermit 1.3.8 (Shearer et al., 2008) as reasoning engine. The tests were done using a PC with an Intel Core 2 Duo 2.8 GHz processor, and 8 GB 800 MHz DDR2 as RAM.

As it can be seen in Figures 3a and 3b, axiomatic policies generation time is of the order of fractions of a second (e.g., 0.15 s with 1000 individuals and 75 policies), and it changes almost linearly in respect to the number of ABOX individuals, and exponentially in respect to the number of rules.

The number of ABOX individuals, as it can be seen in Figure 3c, does not affect the Policy Decision time significantly. In fact, the time for identifying useful individuals in the ABOX represents a small fraction of the total time, while the rest is used for the reasoning algorithms execution.

Figure 3d shows the Policy Decision time in function of the number of rules, with a 2000 individuals ABOX; e.g., the Policy Decision time with 75 rules is 3.65 s. As it can be noticed, response time starts to grow exponentially after 50 rules. It is reasonable to consider that an amount of policies of the order of dozens fits well for many real environments, so, such exponential growth does not seem to represent a major issue at the current state.

In fact, we believe that 75-100 is a realistic amount of policies for different real-life scenarios, e.g., a hospital unit. As it can be seen from Figure 3d, a Policy Decision took about 4-5 seconds in such a case in our tests. That can not be considered a good performance for real-time environments. Anyway, it is reasonable to state that an optimized and compiled code, running on an advanced hardware can lower the Policy Decision time up to a performance that can be considered acceptable, as, e.g., 1 s or lower.

The Policy Decision performances of three different XACML policy engines are presented in (Mourad and Jebbaoui, 2015). They are SBA-XACML engine (Mourad and Jebbaoui, 2015), XEngine (XEngine, 2008) and Sun PDP (SUN XACML, 2004). Even with 400 rules, the Policy Decision time of every engine results under 0.1 s, while we present a Policy Decision time of 3.65 seconds for 75 rules with 2000 individuals. Our performances can be estimated in the order of 100x in time in respect to such standard policy engines. The estimation varies in respect to the specific engine and the number of rules.

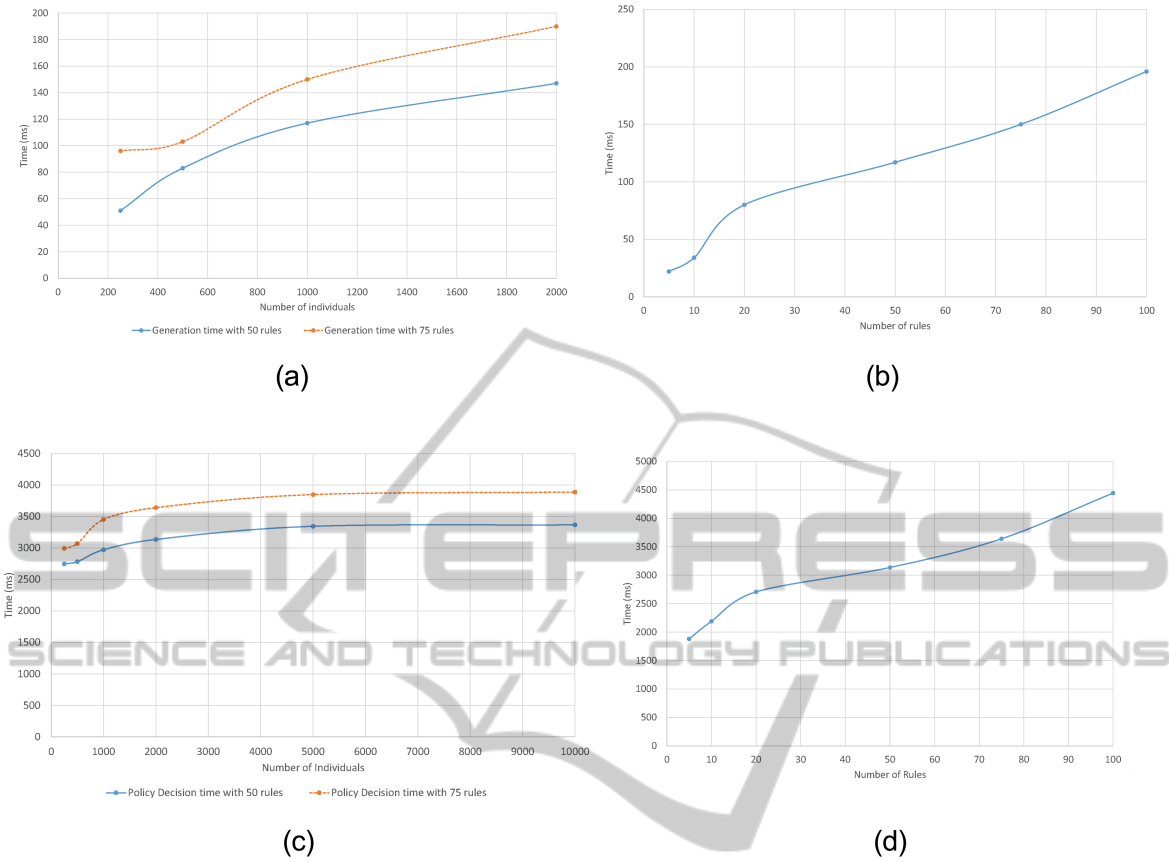

Figure 3: (a) TBOX policies generation time as a function of the number of individuals, with 50 and 75 rules – (b) TBOX policies generation time as a function of the the number of rules, with 1000 individuals – (c) Policy Decision time as a function of the the number of individuals, with 50 and 75 rules – (d) Policy Decision time as a function of the the number of rules, with 2000 individuals.

5 RELATED WORK

(Kolovski et al., 2007) present a formalization of XACML using DDL$^-$, that is a still-decidable extension of DL. Their work is aimed at modeling a generic set of XACML policies using First Order Logic (FOL), in order to easily apply tasks of Policy Harmonization. While their approach covers a wider range of XACML combining algorithms (*permit-overrides*, *deny-overrides*, *first-applicable*), they do not seem to take advantage of FOL expressivity to manage complex policies as our triangular relation Type 5 policies (see Section 3.1.5). Furthermore, their work is not aimed at defining a complete implementation of an XACML framework. In particular, no Policy Decision method is described.

(Finin et al., 2008) presents two different methods for representing RBAC policies with XACML and OWL. The first method consists in representing roles as OWL classes, the second in assigning attributes to specific individuals, in order to identify their roles. We fully adopted the first representation method in our model, describing roles as OWL classes in our Domain TBOX. The second representation method was adopted by (Ferrini and Bertino, 2009) instead, for modeling RBAC access control within XACML. We differ from that approach in the fact that we support ABAC also and we rely on a Domain TBOX for enlarging policy expressivity. Furthermore, they use reasoning for inferring roles and separation of duties only, without implementing a full PDP with OWL and reasoning.

(Ardagna et al., 2009) extend the XACML architecture with more functionalities for credential-based management and privacy control, in the context of the PrimeLife European project (Primelife, 2008). They developed a Policy Decision engine in a complete framework for secure access control, with no use of semantics. While their approach results in better performances, complex tasks as Policy Harmonization

and Policy Explanation seem to be not directly manageable in the environment.

(Paraboschi and Arrigoni Neri, 2013) present a framework for policy specification using OWL, in the context of the PoSecCo European project (Posecco, 2010). PoSecCo project is aimed at proposing new methods and tools for configuring a service landscape in a way to meet security requirements for a generic environment. Policies can be specified at three different levels of abstraction in the framework. Paraboschi and Neri proposes a solution for policy specification in the context of the middle level, that is defined IT Policy level. Policy Harmonization techniques using DL are proposed. No method for Policy Decision is described.

(Mourad and Jebbaoui, 2015) propose an XACML framework implementation, SBA-XACML, for providing an efficient Policy Decision process for web services. Their objective is to provide a real-time decision process for Policy Decision, with better performances than the industrial standard, represented by SUN Policy Decision Point (SUN XACML, 2004). They use set-based algebra for translating and processing XACML policies. Again, while their approach results in efficient performances, complex semantic tasks seems to be not directly manageable.

Between non-XACML-related security frameworks that make use of OWL and reasoning, there are KAoS (Uszok and Bradshaw, 2008) and OWL-POLAR (Sensoy et al., 2012).

KAoS is a collection of software services for the generation of policies that are human expressible and machine enforceable. It presents a user-friendly interface for editing policies, Policy Decision services, specific software for Policy Enforcement. While their approach seems to represent an effective background for complex semantic tasks, their framework is not directly compatible with the XACML standard.

OWL-POLAR is a framework for the expression of permissions and obligations using OWL. Their Policy Decision approach consists in a set of SPARQL-DL queries (Sirin and Parsia, 2007) for a collection of axioms. Policy Harmonization algorithms are provided. Again, while their semantic approach can be a valid background for complex tasks, their framework is not directly compatible with the XACML standard.

6 CONCLUSION AND FUTURE WORK

We presented an implementation of an XACML framework in this paper, based on OWL and reasoning technologies. While policy conversion to DL is executed within a reasonable time, Policy Decision performances are worse than any known XACML engine. 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.

Improvements can be done in the future in the Policy TBOX expressiveness, in order to support more XACML policy combining algorithms (e.g., *permit-overrides*, *first-applicable*), as it is done by (Kolovski et al., 2007). Also, considering temporal constraints for the policies can represent a further advance (see, e.g., (Batsakis et al., 2011)).

Moreover, as stated, a Policy Explanation can be provided together with the response, using OWL Explanation technology (Horridge et al., 2008), as already done by (Marfia, 2014). Also, a Policy Harmonization service based on the OWL policy representation can be developed for the framework, accordingly to what presented in (Paraboschi and Arrigoni Neri, 2013) and (Kolovski et al., 2007).

REFERENCES

Ardagna, C., De Capitani Di Vimercati, S., Neven, G., Paraboschi, S., Pedrini, E., Preiss, F.-S., Samarati, P., and Verdicchio, M. (2011). Advances in Access Control Policies. In Camenisch, J., Fischer-Hubner, S., and Rannenberg, K., editors, *Privacy and Identity Management for Life*, pages 327–341. Springer Berlin Heidelberg.

Ardagna, C., De Capitani di Vimercati, S., Paraboschi, S., Pedrini, E., and Samarati, P. (2009). An XACML-Based Privacy-Centered Access Control System. In *Proc. of the 1st ACM Workshop on Information Security Governance (WISG 2009)*, Chicago, Illinois, USA.

Batsakis, S., Stravoskoufos, K., and Petrakis, E. G. M. (2011). Temporal Reasoning for Supporting Temporal Queries in OWL 2.0. In Konig, A., Dengel, A., Hinkelmann, K., Kise, K., Howlett, R. J., and Jain, L. C., editors, *KES (1)*, volume 6881 of *Lecture Notes in Computer Science*, pages 558–567. Springer.

DL query (2008). DL Query guide - Protégé DLQueryTab. http://protegewiki.stanford.edu/wiki/DLQueryTab.

Ferrini, R. and Bertino, E. (2009). Supporting RBAC with XACML+OWL. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, SACMAT '09, pages 145–154, New York, NY, USA. ACM.

Finin, T., Joshi, A., Kagal, L., Niu, J., Sandhu, R., Winsborough, W. H., and Thuraisingham, B. (2008). ROWLBAC - Representing Role Based Access Control in OWL. In *Proceedings of the 13th Symposium on Access control Models and Technologies*, Estes Park, Colorado, USA. ACM Press.

Fornara, N. and Colombetti, M. (2010). Ontology and Time Evolution of Obligations and Prohibitions Using Semantic Web Technology. In Baldoni, M., Bentahar, J., van Riemsdijk, M., and Lloyd, J., editors, *Declarative Agent Languages and Technologies VII*, volume 5948 of *Lecture Notes in Computer Science*, pages 101–118. Springer Berlin Heidelberg.

Hitzler, P., Krötzsch, M., and Rudolph, S. (2009). *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC. Pages 226–229.

Horridge, M. and Bechhofer, S. (2011). The OWL API: A Java API for OWL Ontologies. *Semant. web*, 2(1):11–21.

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

Horridge, M. and Patel-schneider, P. F. (2008). P.F.: Manchester syntax for OWL 1.1. In *In: OWLED 2008, 4th international workshop OWL: Experiences and Directions (2008) Live Extraction 1223*.

Kolovski, V., Hendler, J., and Parsia, B. (2007). 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. ACM.

López, F. L. Y., Luck, M., and D'Inverno, M. (2006). A Normative Framework for Agent-based Systems. *Comput. Math. Organ. Theory*, 12(2-3):227–250.

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

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

Paraboschi and Arrigoni Neri (2013). *D2.4 - Policy Harmonization and Reasoning*. PoSecCo WP2, Business and IT level policies.

Posecco (2010). PoSecCo - Policy and Security Configuration Management. http://www.posecco.eu/.

Primelife (2008). PrimeLife - Bringing sustainable privacy and identity management to future networks and services. http://primelife.ercim.eu/.

Sensoy, M., Norman, T. J., Vasconcelos, W. W., and Sycara, K. (2012). OWL-POLAR: A Framework for Semantic Policy Representation and Reasoning. *Web Semant.*, 12-13:148–160.

Shearer, R., Motik, B., and Horrocks, I. (2008). HermiT: A Highly-Efficient OWL Reasoner. In Dolbear, C., Ruttenberg, A., and Sattler, U., editors, *OWLED*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Singh, M. P. (1998). An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. *Artificial Intelligence and Law*, 7:97–113.

Sirin, E. and Parsia, B. (2007). SPARQL-DL: SPARQL Query for OWL-DL. In *In 3rd OWL Experiences and Directions Workshop (OWLED-2007*.

SUN XACML (2004). Sun's XACML implementation. http://sunxacml.sourceforge.net/.

SWRL Rules (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL/.

Uszok, A. and Bradshaw, J. M. (2008). Demonstrating Selected W3C Policy Languages Interest Group Use Cases Using the KAoS Policy Services Framework. In *POLICY*, pages 233–234. IEEE Computer Society.

XACML Combining Algorithms (2013). OASIS XACML Version 3.0 Specification, Combining algorithms. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf. page 5.

XACML Data-Flow model (2013). OASIS XACML Version 3.0 Specification, Data-flow model. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf. pages 19–20.

XACML RFC2904 (2000). RFC2904 - AAA Authorization Framework Memo. http://tools.ietf.org/html/rfc2904.

XACML Standard (2013). OASIS eXtensible Access Control Markup Language (XACML). https://www.oasis-open.org/committees/xacml/.

XEngine (2008). XEngine: A Fast and Scalable XACML Policy Evaluation Engine. http://xacmlpdp.sourceforge.net/.

APPENDIX

We present the complete XACML code in this appendix for the example policies mentioned in Section 3. It is shown in Listings 1 and 2.

Listing 1: Example Type 2 policy – XACML specification.

```
<!-- Rule to let medicalConsultant write medicalRegulationDocument -->
<Rule RuleId="Rule2" Effect="Permit">

    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:polimi:names:dbsp:1:function:ontology-id-equal">
            <AttributeValue DataType="urn:polimi:names:dbsp:1:data-type:ontology-id">
              medicalConsultant
            </AttributeValue>
            <SubjectAttributeDesignator AttributeId="urn:polimi:names:dbsp:1:attribute:class" MustBePresent="true"
                DataType="urn:polimi:names:dbsp:1:data-type:ontology-id"/>
          </SubjectMatch>
        </Subject>
      </Subjects>

      <Resources>
        <Resource>
          <ResourceMatch MatchId="urn:polimi:names:dbsp:1:function:ontology-id-equal">
            <AttributeValue DataType="urn:polimi:names:dbsp:1:data-type:ontology-id">
              medicalRegulationDocument
            </AttributeValue>
            <ResourceAttributeDesignator AttributeId="urn:polimi:names:dbsp:1:attribute:class"
                DataType="urn:polimi:names:dbsp:1:data-type:ontology-id"/>
          </ResourceMatch>
        </Resource>
      </Resources>

      <Actions>
        <Action>
          <ActionMatch MatchId="urn:polimi:names:dbsp:1:function:ontology-id-equal">
            <AttributeValue DataType="urn:polimi:names:dbsp:1:data-type:ontology-id">write</AttributeValue>
            <ActionAttributeDesignator DataType="urn:polimi:names:dbsp:1:data-type:ontology-id"
                                       AttributeId="urn:polimi:names:dbsp:1:attribute:id" />
          </ActionMatch>
        </Action>
      </Actions>
    </Target>

  </Rule>
```

Listing 2: Example Type 3 policy – XACML specification.

```
<!-- Rule to deny female access for reading andrology documents -->
<Rule RuleId="Rule3" Effect="Deny">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">F</AttributeValue>
          <SubjectAttributeDesignator AttributeId="urn:polimi:names:dbsp:1:attribute:dataProperty:hasGender"
                                      MustBePresent="true" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>

    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:polimi:names:dbsp:1:function:ontology-id-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            AndrologyDocument
          </AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:polimi:names:dbsp:1:attribute:class"
              DataType="urn:polimi:names:dbsp:1:data-type:ontology-id"/>
        </ResourceMatch>
      </Resource>
    </Resources>

    <Actions>
    <Action>
      <ActionMatch MatchId="urn:polimi:names:dbsp:1:function:ontology-id-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
        <ActionAttributeDesignator DataType="urn:polimi:names:dbsp:1:data-type:ontology-id"
                                   AttributeId="urn:polimi:names:dbsp:1:attribute:id" />
      </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>
```