Journal of Theoretical and Applied Information Technology
31st March 2013. Vol. 49 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A COMPREHENSIVE FRAMEWORK FOR MANAGING CONFLICTS/ANOMALIES BETWEEN XACML POLICIES: MATHEMATICAL BASIS AND ARCHITECTURE

[1]MOHAMED YAHIAOUI, [2]AHMED ZINEDINE, [3]MOSTAFA HARTI,

[1,2,3] UFR INTIC Département d'Informatique Faculté des sciences DharMehraz Fès Maroc

E-mail: [1]yahiaoui.med@gmail.com, [2]ahmedzinedine@yahoo.com , [3]mharti@rocketmail.com

ABSTRACT

In this work we address the problem of detection and resolution of conflicts/anomalies between XACML (*eXtensible Access Control Markup Language*) policies of access control. We mean here by conflict/Anomaly the case where several policies give conflicting answers (deny, allow) to a same access request. Indeed, this problem is foreseeable in access control systems based on policies in general.

We give more attention to the mathematical formalism of the problem. We introduce the notion of the canonical representation of the query space. This is a partition of the query space formed by authorization classes. Each authorization class regroups queries that are intercepted by the same policies. This classification provides a natural way to handle interferences between policy targets (in other words conflicts/anomalies). Then we bring the study of the problem from the whole query space to elements of its canonical representation.

The final result of this work is a Framework for detection and resolution of conflicts/anomalies between XACML policies. This Framework, which is located in the PAP (Policy Administration Point), is responsible for generating a conflicts-free representation from the initially provided policies. This representation is dynamically maintained and updated by the Framework following the addition, deletion or modification of policies.

Keywords: *Access Control; XACML; Policy; Anomaly; Conflict; Anomaly Detection And Resolution; FIA Algebra; Canonical Representation*

1. INTRODUCTION

The control of access to resources of a computer system is often implemented through a set of rules and policies that reflect the desired level of restriction. These rules and policies are written in specific languages.

Among these languages there is the *OASIS* standard *eXtensible Access Control Markup Language* (*XACML*). *XACML* is mainly based on attributes. These are associated with users, resources, actions and environment and they constitute the inputs to the decision system. On the basis of these attributes, the system decides when a given user can perform a given action on a given resource.

One of the problems of policy-based approach is how to ensure coherence between policies. Indeed, the situation where multiple policies are applicable to the same query is possible. We may end up in a conflict scenario where some policies allow access and others deny it.

This topic has taken considerable attention lately. Several approaches have been proposed to solve the problem of detection and resolution of conflicts/anomalies in access control systems. In [2] the authors introduced a technique of segmentation to identify anomalies and to derive resolutions based on XACML standard *combining algorithms*. The paper [3] proposed a unified framework for policy analysis, detection and resolution of anomalies. The framework is based on a generic approach to capture common semantics of policies. The authors of [5] introduced a formal model that is compatible with the *Alloy* language for specifying access control policies and then they used this model for automatic detection of anomalies using the Alloy analyzer.

However, the following points require further investigation:

- How to represent in a clear and unified manner the conflicts/anomalies in an access control system. This representation should be understandable by humans, and it should be a

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

support for decision making about conflict resolution.

- The dynamic aspect of the representation of conflicts/ anomalies: we mean here the monitoring of the impact of adding, modifying and deleting policies on the representation of conflicts/anomalies in the access control system. The administrator must have at any moment a clear view on the situation of conflict/anomalies in the system.
- The integration of the component of detection and resolution of conflicts/anomalies in a comprehensive system of access control. For example in the case of XACML, the integration consists in defining and then locating the component of detection and resolution of conflicts/anomalies relative to other standard components: PEP, PDP, PIP and PAP.

In this work we propose a comprehensive Framework for conflicts/anomalies management. We give special attention to the mathematical formalism of the problem. To do this, and in order to express formally the notion of conflict between policies and interference between different authorization spaces, we introduced an equivalence relation in the query space. We show that the quotient space with respect to this relation is exactly what we call "*canonical representation of the query space*". From this canonical representation, we derive a conflict-free representation of the policies.

Indeed, the proposed representation is a segmentation of the query space into subsets (authorization classes). Each class can be seen as the target of a policy (in a future conflict-free system). It only remains to calculate the decision of this policy by combining different active policies on each authorization class. To this end, we use *policy integration expressions* introduced by the FIA algebra [1]. These expressions provide a mechanism richer than the standard XACML combining algorithms. Then we give a method for calculating the projection of the FIA policy integration expression on the different authorization classes. This allows bringing the study of conflict resolution from the query space to its canonical representation.

Also, we study the impact of the dynamic aspect of the policy repository. So we study the evolution of the canonical representation of the query space following the addition, modification and deletion of policies from the policy repository. In fact, this dynamic aspect of the policy repository is one of the concerns of the access control system administration that requires suitable decision-making support. Hence the importance of integrating the canonical representation as a part of a Framework for conflict detection and resolution in XACML access control systems.

We also show the validity of the Framework and the feasibility of implementation by detailing the most important algorithms. The user orientation is one of the strengths of this Framework. Indeed, the proposed architecture promotes interactions between the access control system and its administrators.

The rest of this paper is organized as follows: in section 2 we give some general notions that we estimate necessary to understand the paper, especially on the XACML language and the FIA algebra. We prove in section 3 the mathematical basis of our approach and finally, in section 4, we describe the architecture of the Framework and we detailed the most important algorithms for the implementation.

2. PRELIMINARIES

2.1. Extensible access control markup language (xacml)

XACML, as defined in the *OASIS* standard [6], allows the implementation of an access control system compatible with service-oriented environments (*SOA*). The *XACML* standard provides:

- An XML language for expressing access control policies,
- A language to express authorization requests,
- A language to express authorization decisions (responses),
- An architecture that defines the main components of an implementation and an information-flow model.

2.1.1. Policy

The access control policies are used to define system behavior against access requests. Indeed, a policy is a set of rules on the attributes of subjects, resources, actions and environment. These rules are combined using logical operators to determine the rights of users on system resources.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

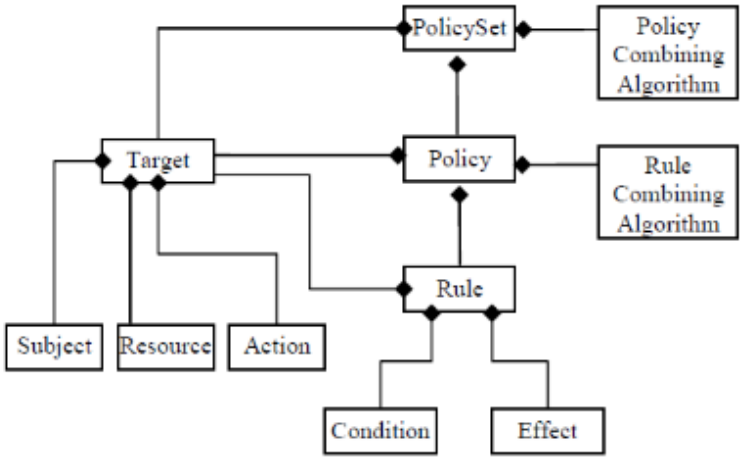

Figure 1. XACML data-flow diagram [6]

2.1.2. Request

XACML requests are written in an XML encoding that allows expressing the attributes of the user, the requested resource, the desired action and other environmental information. The requests are sent to the PDP (see next section). This last extracts the attributes of the request and compares them with the targets of policies to determine XACML policies that are applicable to this request and then determines the decision.

The XACML policy language defines three different decisions: (i) Permit, (ii) Deny, (iii) Not applicable.

2.1.3. XACML Data-flow Model

The XACML data-flow model defines a modular and distributed architecture of the access control system. It defines the various components and their roles. It defines also the possible exchanges of messages between these components and the structure of these messages. The figure below illustrates this architecture.

PAP: The Policy Administration Point is the point of administration of policies. It allows administrators to maintain the policy repository.

PEP: The Policy Enforcement Point is the component that receives access requests to resources. It is responsible for transforming these requests to XACML format, and then it transmits them to the PDP. Depending on the response of the PDP, the PEP allows or denies access to the resource. In case of error a feedback can be raised.

PDP: The Policy Decision Point is the component that handles XACML requests received from the PEP and then gives the appropriate XACML response. This is an authorization decision (Permit/Deny) based on the policies stored in the PAP.

Journal of Theoretical and Applied Information Technology
31st March 2013. Vol. 49 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

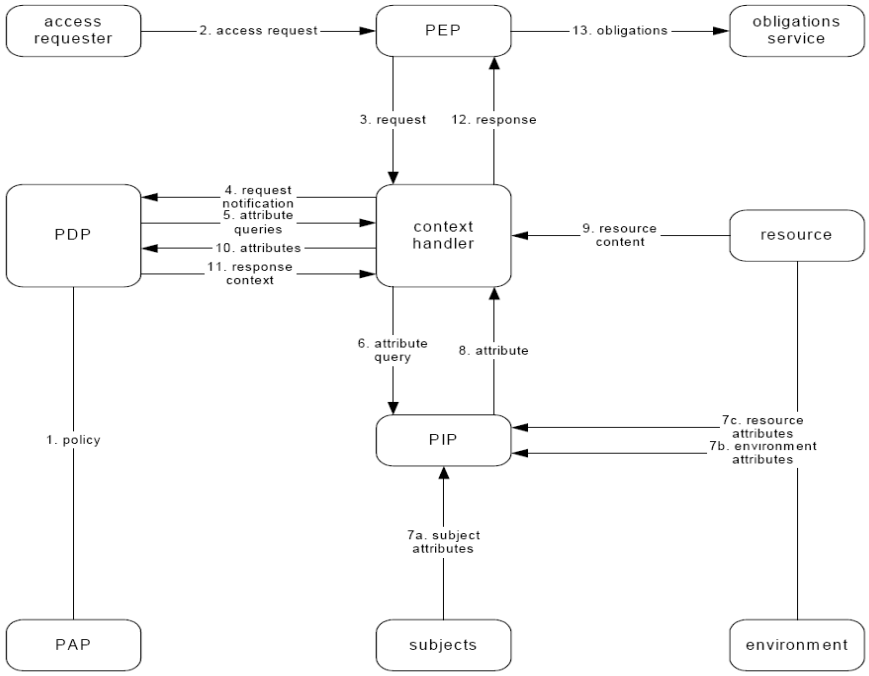

Figure 2. XACML data-flow diagram [6]

2.2.The Fine-Grained Integration Algebra (FIA)[1]

The XACML standard policy combining algorithms allow the implementation of strategies for policy integration. But they do not give enough flexibility to support the requirements of applications and involved parties.

To resolve this problem the FIA algebra was introduced in [1]. It provides an efficient mechanism for policy integration. It can support a strategy of flexible and granular policy integration. Among the advantages of the FIA algebra we quote the following:

- It supports policies expressed in rich languages like XACML;
- It allows to solve the problems related to the integration of heterogeneous and fine-grained access control policies;
- This algebra is able to support the specification of a wide variety of integration constraints;
- The algebra is highly expressive.

We give below a quick overview of the FIA algebra and its main concepts.

2.2.1. The semantics of queries and policies

The formalism of queries is based on the finite set '*A*' of attribute names. *A* is composed of attributes characterizing the subjects, resources, actions and environment. Each element of '*A*' has a domain, denoted dom(*a*), consisting of all possible values of the attribute *a*.

Definition 1 A request r is defined as follows: $r \equiv \{(a_1, v_1), (a_2, v_2), \cdots, (a_k, v_k)\}$, where a_1, a_2, ..., a_k are attribute names, and $v_i \in dom(a_i)$ $(1 \leq i \leq k)$.

Definition 2 let P be an access control policy. The semantics of P is defined as a 2-uplet $\langle R_Y^P, R_N^P \rangle$, where R_Y^P (resp. R_N^P) is the set of allowed (resp. denied) requests. Note that $R_Y^P \cap R_N^P = \emptyset$.

2.2.2. The operators of the FIA algebra

The fine-grained integration algebra (FIA) is given by $\langle \sum, P_Y, P_N, +, \&, \neg, \prod_{dc} \rangle$, where $\sum$ is a vocabulary of attribute names and their domains, P_Y and P_N two policy constants, + and & are two binary operators, and ¬ and $\prod_{dc}$ are two unary ones.

To illustrate the operators and constants of the FIA algebra let P_1, P_2 and P_I be three policies such that

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$P_1 \equiv \langle R_Y^{P1}, R_N^{P1} \rangle$,
$P_2 \equiv \langle R_Y^{P2}, R_N^{P2} \rangle$ and $P_I \equiv \langle R_Y^{PI}, R_N^{PI} \rangle$.

Permit policy (P_Y). P_Y is a policy constant that permits everything.

Deny policy (P_N). P_N is a policy constant that denies everything.

Addition (+) : the addition of two Policies P1 and P2 is a policy that allows all requests authorized by P1 or P2, and refuses requests that are denied by one of the two policies and not authorized by the other. More precisely:

$P_I = P_1 + P_2 \Leftrightarrow R_Y^{PI} = R_Y^{P1} \cup R_Y^{P2} \wedge R_N^{PI} = (R_N^{P1} \setminus R_Y^{P2}) \cup (R_N^{P2} \setminus R_Y^{P1})$

Intersection (&):

$P_I = P_1 \,\&\, P_2 \Leftrightarrow R_Y^{PI} = R_Y^{P1} \cap R_Y^{P2} \wedge R_N^{PI} = R_N^{P1} \cap R_N^{P2}$

Negation (¬): $P_I = \neg P \Leftrightarrow R_Y^{PI} = R_N^{P} \wedge R_N^{PI} = R_Y^{P}$.

Domain projection ($\prod_{dc}$):

Definition 3 A domain constraint dc takes the form $\{(a_1, plage_1), (a_2, range_2), \ldots, (a_k, range_k)\}$, where $a_1, a_2, \ldots a_k$ are attribute names, and $range_i$ $(1 \leq i \leq k)$ are sets of values from the vocabulary $\sum$. Given a request $r = \{(a_{r1}, v_{r1}), (a_{r2}, v_{r2}), \cdots, (a_{rm}, v_{rm})\}$. We say that r satisfies dc if the following condition holds: for each $(a_{rj}, v_{rj}) \in r$ $(1 \leq j \leq m)$, if there exists $(a_{rj}, range_i) \in dc$, then $v_{rj} \in range_i$.

The semantics of $(\prod_{dc})$ is given by

$P_I = \prod_{dc} (P) \Leftrightarrow R_Y^{PI} = \{r \mid r \in R_Y^P$ and r satisfies dc $\}$, $R_N^{PI} = \{r \mid r \in R_N^P$ and r satisfies dc $\}$.

Not-applicable policy (P_{NA}) P_{NA} is a policy constant that is not applicable for every request. It is defined as $P_{NA} = P_Y \,\&\, P_N$.

Effect projection (Π_Y and Π_N) Π_Y (P) restricts the policy P to the requests allowed by it. It is defined as: Π_Y (P) = P &P_Y. Similarly, Π_N(P) restricts the policy P to the requests denied by it; it is defined as $\Pi_N(P) = P \,\&P_N$.

Subtraction (−) Given two policies P_1 and P_2. The subtraction operator is defined as: $P_1 - P_2 = (P_Y \,\&(\neg(\neg P_1 + P_2 + \neg P_2))) + (PN \,\&(P_1 + P_2 + \neg P_2))$.

Precedence (▷): Given two policies P_1 and P_2, $P_1 \triangleright P_2$ is the policy that gives the same decision as P_1 for all queries applicable to P_1 and gives the same decision as P_2 for all other queries. It can be easily shown that $P_1 \triangleright P_2 = P_1 + (P_2 - P_1)$.

2.2.3. The expressions of the FIA algebra

A FIA expression is recursively defined as follows:

- If P is a policy, then P is a FIA expression.
- If f1 and f2 are FIA expressions so are (f1) + (f2), (f1)&(f2), and ¬(f1).
- If f is a FIA expression and dc is a domain constraint then Π_{dc}(f) is a FIA expression.

The FIA algebra can express not only the standard algorithms for combining XACML policies, but also other more complex integration constraints. We give below some examples:

Let P_1, P_2,..., P_n be access control policies. The standard algorithms *permit-overrides*, *Deny-overrides*, *first-one-applicable* and *only-one-applicable* can be expressed respectively by the following FIA expressions: $P_1+P_2+ \ldots +P_n$, $\neg((\neg P_1) + (\neg P_2) + \ldots + (\neg P_n))$, $P_1 \triangleright P_2 \triangleright \ldots \triangleright P_n$ and $(P_1 - P_2 - P_3 - \ldots - P_n) + (P_2 - P_1 - P_3 - \ldots - P_n) + \ldots + (P_n - P_1 - P_2 - \ldots - P_{n-1})$.

Now we give an example of a more specific and more complex integration constraint. Let P_1, P_2 and P_3 be three policies. We aim to express the following integration constraint: For a given query r_1, if P_1 allows r_1 then the final decision will be that of the policy P_2, if not the final decision will be that of P_3. This integration constraint can be expressed by the following FIA expression:

$\Pi_Y(P_1 \,\&\, P_2) + \Pi_N(\neg P_1 \,\&\, P_2)) + \Pi_Y(\neg P_1 \,\&\, P_3) + \Pi_N(P_1 \,\&\, P_3))$

2.3. Representation of XACML policies by compound Boolean expressions over request attributes [1] :

XACML policies can be transformed into compound Boolean expressions over request attributes where each compound Boolean expression consists of atomic Boolean expressions (AE) combined using the logical operations "∨" and "∧". Atomic Boolean expressions that appear in most policies fall into one of two categories:

i. Constraints of equality to a constant: $a = c$, $a \neq c$, where a is an attribute name and c is a constant.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ii. Constraints of inequality to a constant: $c_1\ rel_1\ a\ rel_2\ c_2$, where a is an attribute name, c_1 and c_2 are two constants and $rel_1, rel_2 \in \{<, \leq, >, \geq\}$.

So a policy can be expressed as follows:

$$P(x) = \begin{cases} Y, & if\ f_1(x_1, x_2, \dots, x_n) \\ N, & if\ f_2(x_1, x_2, \dots, x_n) \end{cases} \square\square$$

Where x is a query and $x_1, x_2 \dots x_n$ are the values of attribute names of the query x and f_1 and f_2 are two Boolean expressions.

3. MATHEMATICAL BASIS OF THE FRAMEWORK

3.1. Motivation example

We illustrate the issues of conflicts/anomalies between policies through an example in a geographic information system of a city that is open to General Public and Town Planners groups. The system consists of three layers: buildings, roads and population. The access control requirements are as follows:

- The General Public group has read access to the road layer.
- The group Town Planners accesses all levels for reading and writing.

The implementation of these requirements is made by the following policies:

P_1:

(Role = General Public, resource = all features of road layer, action = read, decision = Permit)

(Role = General Public, resource = all features of all layers, action = write, decision = Deny)

P_2:

(Role = Town Planners, resource = all features of all layers, decision = Permit)

Suppose now that the administrator of access control system decides to ban access to features of layers located near a sensitive infrastructure (the aim is not to reveal the presence of this last). This requirement can be implemented via the following policy:

P_3: (Resources = all features of all layers, feature.geom $\cap$ polygon1 $\neq \phi$, decision = Deny), where Polygon1 is a polygon containing the sensitive infrastructure.

It is obvious that P_3 is in conflicts/anomalies with existing policies.

To eliminate these conflicts/anomalies, the administrator of the access control system has two solutions:

i. Use of standard policy combining algorithms.

ii. Redesign of all existing policies to build a system without conflicts/anomalies.

It is clear that the last solution is not practical, because of existing policies may be quite numerous and complex to the point where the redesign is almost impossible. While the first option, the following question is obvious: what policy combining algorithm should be used to meet the above requirements? The following example shows that, sometimes, none of the standard combining algorithms produces the intended result.

3.1.1. Case study

Let Polygon2 be another polygon such that Polygon2 $\cap$ Polygon1 = ϕ. Consider the following requests:

R_1: (role = General Public, resource = road layer, action = read, extent = Polygon2). It is clear that this request will be intercepted only by policy P_1.

R_2: (role = Town Planners, resource = building layer, action = read, layer extent = Polygon2). This request will be intercepted only by policy P_2.

R_3: (role = General Public, resource = population layer, action = read, extent = Polygon1). This request will be intercepted only by policy P3.

R_4: (role = Town Planners, resource = road layer, action = write, extent = Polygon2). This request will be intercepted by policies P_1 and P_2,

R_5: (role = General Public, resource = road layer, action = read, extent = polygon1). It is clear that this request will be intercepted by policies P_1 and P_3,

R_6: (role = Town Planners, resource = building layer, extent = Polygon1). This request will be intercepted by policies P_2 and P_3,

R_7: (role = Town Planners, resource = road layer, action = read, extent = Polygon1) It is clear that this request will be intercepted by policies P_1, P_2 and P_3.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

R	*P1*	*P2*	*P3*	*intended Result*	*permit-overrides*	*Deny-overrides*	*first-one-applicable*	*only-one-applicable*
R1	Y	NA	NA	Y	Y	Y	Y	Y
R2	NA	Y	NA	Y	Y	Y	Y	Y
R_3	NA	NA	N	N	N	N	NA	N
R_4	N	Y	NA	Y	Y	N	N	Y
R_5	Y	NA	N	N	Y	N	Y	NA
R_6	NA	Y	N	N	Y	N	NA	NA
R_7	Y	Y	N	N	Y	N	Y	NA

The previous table shows that none of the traditional combining algorithms can integrate all three policies by producing the intended decisions that meets the previously specified requirements. Indeed each request in the previous example reflects a particular conflict situation. Each situation is characterized by the set of policies applicable to the request. But the standard combining algorithms lack the necessary mechanisms to customize the resolution of conflict.

In order to build an efficient solution for dealing with conflicts/anomalies, we include some requirements derived from the above example:

- A method of segmenting the query space into zones of conflicts/anomalies, able to give us an advanced understanding of policy interference.
- Tools and algorithms able to identify policy interference zones by a set of conditions on the attributes of subjects, resources and environment.
- The ability to use more efficient algorithms for policy integration.

The solution we propose to meet this needs is to implement a Framework for detection and resolution of anomalies. This framework must meet the following specifications:

- The Framework will be an extension of the PAP.
- It automates the detection of conflicts/anomalies and provide a clear representation,
- It updates the list of conflicts/anomalies after the addition, deletion and modification of policy,
- It Provides automatic resolutions based on customized combination algorithms (expressed using FIA expressions),
- It provides the possibility for administrators to manually implement the resolutions,
- It makes available to the PDP an error-free representation of the set of policies. This representation must be updated automatically following addition or deletion of policies.

3.2. Definitions and mathematical formalism

Definition 4 [2] (*Authorization Space*) let P be a policy. The authorization space of P is defined as the set of requests Q_p for which the policy P is applicable, i.e. $Q_P = R_Y^P \cup R_N^P$. It can be given by:

$Q_p = \{x \in R_\Sigma / f(x_1, x_2, \dots, x_n)\}$, where $f = f_1 \vee f_2$ is the Boolean expression that defines P and x_1, x_2 … x_n are the attributes of x.

An anomaly is defined as a situation where several policies intercept a same query to give access decisions (allow / deny). If, moreover, these decisions are contradictory, this is said to be a conflict. This situation is interpreted by the intersection of several authorization spaces.

Definition 5 (*conflict/anomaly*) let P_1 and P_2 be two policies. We say that there is a conflict/anomaly between P_1 and P_2 if $Qp_1 \cap Qp_2$ is not empty.

In the following we propose a method to classify queries so that derived classes include queries that can be processed by the system in similar ways. A class corresponds to the queries intercepted by exactly the same policies. Then we give the following definition of an authorization class:

Definition 6 (*authorization class*) let E' ={P'_1, … ,P'_m} be a given subset of policies. The authorization class of E' is defined as the set of all requests to which P'_1, … ,P'_m -and only P'_1, … ,P'_m- are applicable. We note this set *AC(E')*.

3.2.1. The canonical representation of the query space

Let E be a finite set of policies, $\sum$ is a vocabulary of attribute names and their domains. The query Space R_Σ has a canonical representation that reflects the behavior of policies against requests. We introduce below this canonical representation.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Let r_1 and r_2 be two requests of R_Σ. We denote by $\Re$ the relationship in R_Σ defined by $r_1 \Re r_2$ if, and only if, for each policy P in E, $r_1 \in Q_P \Leftrightarrow r_2 \in Q_P$. In other words, the set of applicable policies to r_1 is exactly the set of those applicable to r_2.

One can easily show that the relationship «$\Re$» is an equivalence relation. We denote by $\overline{r}^{E}$ the equivalence class of the request r relatively to E.

Definition 7 The quotient space $R_\Sigma/\Re$ that is constituted of all equivalence classes is called *the canonical representation* of R_Σ.

Since the set E is finite, the set $\wp(E)$ of subsets of E is also finite. And since by definition each equivalence class $\overline{r}^{E}$ is associated with an element of $\wp(E)$ (we denote it $P(\overline{r}^{E})$) then the set of all equivalence classes is finite. Therefore there exist $r_1, \ldots, r_n \in R_\Sigma$ such that $(\overline{r_i}^{E})_{(1 \le i \le n)}$ is a partition of R_Σ. From the well-known properties of equivalence relations this partition is unique. Hence the following result:

Proposition 1 There exists a finite set of requests $r_1, \ldots, r_n \in R_\Sigma$ such that $(\overline{r_i}^{E})_{(1 \le i \le n)}$ constitute the canonical representation of R_Σ.

Let E' $=\{P'_1, \ldots, P'_m\}$ be a subset of the set E of all policies in the access control system. Then for each request r in AC(E') (see definition 6) we have $AC(E') = \overline{r}^{E}$.

The following result allows calculating, more explicitly, authorization classes using Boolean expressions.

Proposition 2 Let $E = \{P_1, P_2, \ldots, P_n\}$ be the set of all policies in the access control system. For each P in E denote f_P the Boolean expression defining the authorization space of the policy P (see [1]) and let $P(\overline{r}^{E}) = b$. One can easily show that:

$$\overline{r}^{E} = \{r \in R_\Sigma / (\bigwedge_{P \in b} f_P(x_1, x_2, \ldots, x_m)) \wedge (\bigwedge_{p \in b^c} \neg f_P(x_1, x_2, \ldots, x_m))\}$$

3.2.2. Illustration example

Let $E = \{P_1, P_2, P_3, P_4\}$, where P_1, P_2, P_3 and P_4 are XACML policies and r is a request.

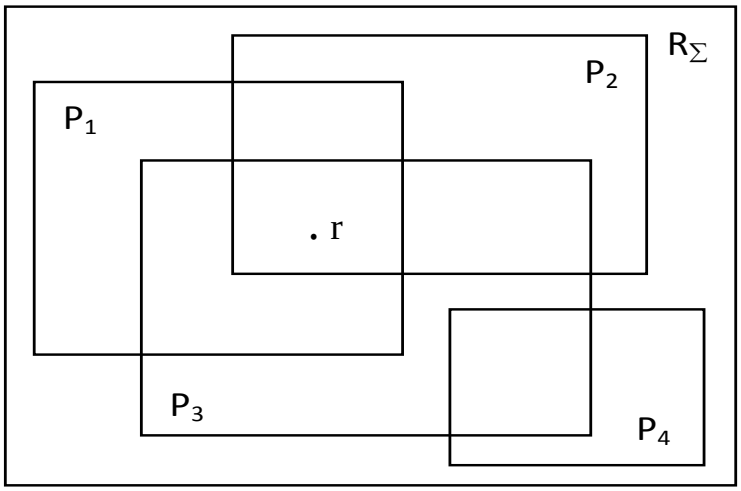

Figure 3. Illustration Of The Authorization Spaces In R_Σ

In the figure above R_Σ represents the query space. Each rectangle represents requests that are intercepted by a given policy. For example the rectangle P_1 surrounds all requests to which the policy P_1 applies. Note that the request r is intercepted by P_1 and P_3 (and only by P_1 and P_3).

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

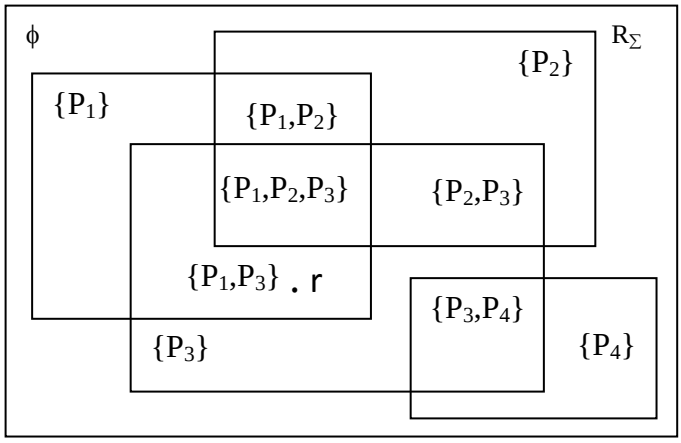

Figure 4. Illustration Of The Canonical Representation Of R_Σ

In the figure above R_Σ is divided into 10 zones (deduced from Figure 5): $\{P_1\}$, $\{P_2\}$, $\{P_3\}$, $\{P_4\}$, $\{P_1,P_2\}$, $\{P_1,P_3\}$, $\{P_2, P_3\}$, $\{ P_3,P_4\}$, $\{P_1,P_2,P_3 \}$ and ϕ.

Each zone represents all requests intercepted exactly by a given set of policies. For example the zone $\{P_1, P_3\}$ represents requests intercepted by, and only by, P_1, P_3. $\{P_4\}$ represents requests intercepted by, and only by, P_4. The zone denoted ϕ represents queries that do not match any policy.

Indeed, each zone corresponds to an equivalence class of the canonical representation of R_Σ. In the example above the zone $\{P_1,P_3\}$ is the equivalence class of r.

We note that these classes correspond to possible interferences between the policies of E. We also note that.

- By definition each authorization class $\overline{r^E}$ is generated by an element of $\wp(E)$. Denote this element $P(\overline{r^E})$. (in the previous figure $P(\overline{r^E})=\{P_1,P_3\}$),
- There exists an equivalence class represented by ϕ. It includes queries that belong to no authorization space.
- We note that an element of $\wp(E)$ may not correspond to an authorization class. For example $\{P_1,P_4\}$, $\{P_2,P_4\}$, $\{P_1, P_3,P_4\}$ et $\{P_1,P_2,P_3,P_4\}$.
- One can deduce from the remarks above that there is a subset B of $\wp(E)$ such that for each element $r \in R_\Sigma$, there exists $b \in B$ and $P(\overline{r^E})= b$.
- Authorization classes $\overline{r^E}$ with $card(P(\overline{r^E}))$ equals to 1 or 0 are not included in any conflict/anomaly.

In the following we study the impact of changing the initially provided set of policies on the canonical representation of the query space.

3.3. Impact of the modification of initialy provided policy set on the canonical representation

3.3.1. Policy addition

When adding a new policy to the policy repository, the canonical representation must be updated. Here we give some necessary results to this end:

Proposition 3 Let $E = \{P_1,P_2, \ldots, P_n\}$ and $E' = E \cup \{P_{n+1}\}$. Then $\forall r \in R_\Sigma$:

- $\overline{r^E} = \overline{r^{E'}} \cup \overline{r_1^{E'}}$ with $r_1 \in (\overline{r^E} \setminus \overline{r^{E'}})$ if $(\overline{r^E} \setminus \overline{r^{E'}}) \neq \phi$
- or $\overline{r^E} = \overline{r^{E'}}$ otherwise.

In other words, an element of the canonical representation of R_Σ in E is equal either:

- to the union of two elements of the canonical representation of R_Σ in $E \cup \{P_{n+1}\}$ or,
- to an element of the canonical representation of R_Σ in $E \cup \{P_{n+1}\}$,

Proof

Let $P(\overline{r^E}) = \{P'_1, \ldots ,P'_m\}$. By definition $\forall r \in \overline{r^E}$ $\{P'_1, \ldots ,P'_m\}$ are the only applicable policies to r in E.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

To determine authorization classes in E' we must study the applicability of P_{n+1} to requests in $\overline{r^E}$. Indeed, we have three cases depending on whether the policy P_{n+1} is applicable or not to requests in the authorization class $\overline{r^E}$:

Case 1: P_{n+1} is applicable to all requests in $\overline{r^E}$

Let r' $\in$ $\overline{r^E}$. Then {P'$_1$, … ,P'$_m$, P_{n+1}} are applicable to r'. To prove that r' $\in$ $\overline{r^{E'}}$ it suffices to prove that {P'$_1$, … ,P'$_m$, P_{n+1}} are the only applicable policies to r' in E'.

Suppose that there exists P_j in E$\cup${P_{n+1}}\{P'$_1$, … ,P'$_m$, P_{n+1}} such that P_j is applicable to r '. Then there exists P_j in E\{P'$_1$, … ,P'$_m$} such that P_j is applicable to r'. This contradicts the assumption r' $\in$ $\overline{r^E}$. Therefore r'$\in$ $\overline{r^{E'}}$.

Hence $\overline{r^E}$ = $\overline{r^{E'}}$.

Case 2 : P_{n+1} is not applicable to any queries in $\overline{r^E}$

In the same way as case 1 we can show that $\overline{r^E}$ = $\overline{r^{E'}}$.

Case 3: P_{n+1} is applicable to a strict subset of $\overline{r^E}$

Let r_1, r_2 $\in$ $\overline{r^E}$ such that P_{n+1} is applicable to r_1 and not applicable to r_2. Then P($\overline{r_2^{E'}}$) = {P'$_1$, … ,P'$_m$} and P($\overline{r_1^{E'}}$) = {P'$_1$, … ,P'$_m$, P_{n+1}}. It is clear that $\overline{r_1^{E'}} \cup \overline{r_2^{E'}} \subseteq \overline{r^E}$.

Inversely, let r' $\in$ $\overline{r^E}$. Then {P'$_1$, … ,P'$_m$} are applicable to r'. If P_{n+1} is applicable to r' then r' $\in$ $\overline{r_1^{E'}}$ otherwise r' $\in$ $\overline{r_2^{E'}}$. Therefore $\overline{r^E} \subseteq \overline{r_1^{E'}} \cup \overline{r_2^{E'}}$.

Proposition 4 let E = {P_1,P_2, …, P_n}, P_i and P_j are two policies of E, f_i (resp. f_j) is the atomic Boolean expression of P_i (resp. of P_j) on the attribute name *a*. Then, if for each value v in the domain of *a*, $f_i(v) \wedge f_j(v)$ = false then P_1 and P_2 can not belong to the same policy set defining an element of the canonical representation.

The proof is obvious.

3.3.2. Deleting a policy

The canonical representation must also be updated after the deletion of a policy from the policy repository.

Let E = {P_1,P_2, …, P_n} and E' = E $\cup$ P_{n+1} two policy sets. Let CR (resp. CR') the canonical representation of R_Σ with respect to E (resp. E'). One can deduce from Proposition 3 above that to reconstruct CR from CR', it suffices to combine each two elements of CR that have respective applicable policies of the form {P'$_1$, … ,P'$_m$, P_{n+1}} and {P'$_1$, … ,P'$_m$}.

Note that we did not deal here with the policy modification because it is equivalent to deleting a policy and then replacing it by a new one.

3.3.3. Calculation of the policy integration expression on canonical representation elements

In the above we have segmented the query space into subsets (authorization classes). Each class can be seen as the target of a policy (in a future conflict-free system). It remains to calculate the decision of this policy by combining different active policies on each authorization class.

Policy integration expressions introduced by the FIA algebra [1] provide a mechanism richer than the standard XACML combining algorithms. In the next proposition we calculate the projection of the FIA policy integration expression on the different authorization classes (that we call *the resultant policy integration expression*). The final goal is to bring the study of conflict resolution from R_Σ to its canonical representation.

Proposition 5 Let E = {P_1,P_2, …, P_n} be a set of policies and dc a domain constraint. Let $(\prod_{dc}(E)$ be the set of policy projections on dc defined as follows : $\prod_{dc}(E)$ = {$\prod_{dc}(P)$| P $\in$E}. let f be a FIA policy integration expression in E. Then there exists a policy integration expression f_{dc} in $\prod_{dc}(E)$ such that $\prod_{dc}(f(P_1,P_2, …, P_n)) = f_{dc}(\prod_{dc}(P_1), \prod_{dc}(P_2), \cdots, \prod_{dc}(P_n))$

Proof

According to *the "minimum set of operators" theorem* (see [1]), it suffices to prove the proposition for the following operators {¬ , P_Y, &, +, Π_{dC}}.

Suppose that f consists of a single monomial then f can be written in one of the following formats:

$f(P_1,P_2, …, P_n) = P_i$,

$f(P_1,P_2, …, P_n) = \neg P_i$,

$f(P_1,P_2, …, P_n) = \prod_{dc1}(P_i)$,

$f(P_1,P_2, …, P_n) = P_Y$

Then f_{dc} will be defined respectively as follows:

$f_{dc}(\prod_{dc}(P_1), \prod_{dc}(P_2), \cdots, \prod_{dc}(P_n)) = \prod_{dc}(P_i)$,

$f(\prod_{dc}(P_1), \prod_{dc}(P_2), \cdots, \prod_{dc}(P_n)) = \neg\prod_{dc}(P_i)$,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$f(\prod_{dc}(P_1), \prod_{dc}(P_2), \dots, \prod_{dc}(P_n)) = \prod_{dc1}(\prod_{dc}(P_i))$,

$f(\prod_{dc}(P_1), \prod_{dc}(P_2), \dots, \prod_{dc}(P_n)) = P_Y$.

It is clear that in all cases:

$\prod_{dc}(f(P_1, P2, \dots, P_n)) = f_{dc}(\prod_{dc}(P_1), \prod_{dc}(P_2), \dots, \prod_{dc}(P_n))$.

Suppose now that the result is true for an expression of *m* monomials and show that this remains true for an expression of a *(m+1)* monomials:

Let f be an expression of *(m+1)* monomials. Then there exists an expression f' of *m* monomials such that:

$f(P_1,P_2, \dots, P_m, P_{m+1}) = f'(P_1,P_2, \dots, P_m)$ <binary_opetrator> (<unary_ opetrator > P_{m+1})

We define f_{dc} as follows $f_{dc} = f'_{dc}$ <binary_opetrator> (<unary_ opetrator > $\prod_{dc}(P_{m+1})$)

$\prod_{dc}(f(P_1,P_2, \dots, P_m, P_{m+1})) = \prod_{dc}(f'(P_1,P_2, \dots, P_m)$ <binary_opetrator > (<unary_ opetrator > P_{m+1}))

$\prod_{dc}(f(P_1,P_2, \dots, P_m, P_{m+1})) = \prod_{dc}(f'(P_1,P_2, \dots, P_m))$ <binary_opetrator > $\prod_{dc}$ (<unary_ opetrator > P_{m+1})

According to the recurrence hypothesis : $\prod_{dc}(f'(P_1,P_2, \dots, P_m)) = f'_{dc}(\prod_{dc}(P_1), \prod_{dc}(P_2), \dots, \prod_{dc}(P_m))$

$\prod_{dc}(f(P_1,P_2, \dots, P_m, P_{m+1})) = f'_{dc}(\prod_{dc}(P_1), \prod_{dc}(P_2), \dots, \prod_{dc}(P_m))$ < binary_opetrator > (<unary_ opetrator> $\prod_{dc}(P_{m+1})$)

$\prod_{dc}(f(P_1,P_2, \dots, P_m, P_{m+1})) = f_{dc}(\prod_{dc}(P_1), \prod_{dc}(P_2), \dots, \prod_{dc}(P_m), \prod_{dc}(P_{m+1}))$.

4. DESCRIPTION AND IMPLEMENTATION OF THE FRAMEWORK

4.1. Architecture of the Framework

Our Framework is designed for access control systems based on XACML. This type of systems is characterized by a finite number of policies. The Framework supports a mechanism for policy integration based on FIA expressions.

The Framework of anomalies detection and resolution will be an extension of the PAP component which is a part of the standard architecture of XACML. To illustrate the integration of this Framework into the XACML architecture, we present in the figure below the PAP architecture with and without the component of detection and resolution of anomalies.

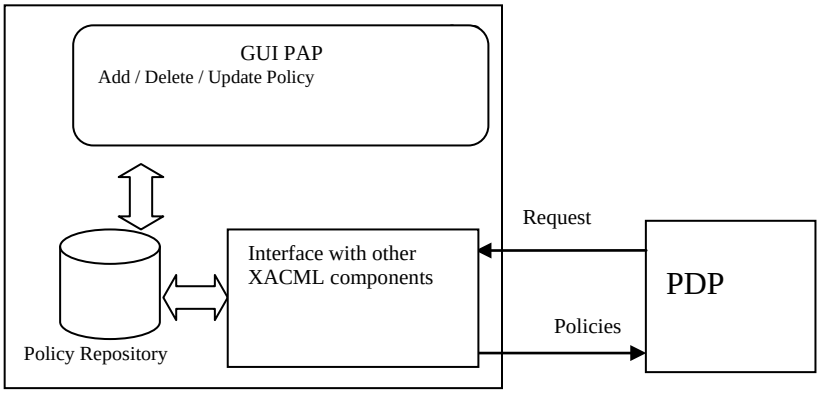

Figure 5. PAP Standard Architecture

Journal of Theoretical and Applied Information Technology
31st March 2013. Vol. 49 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

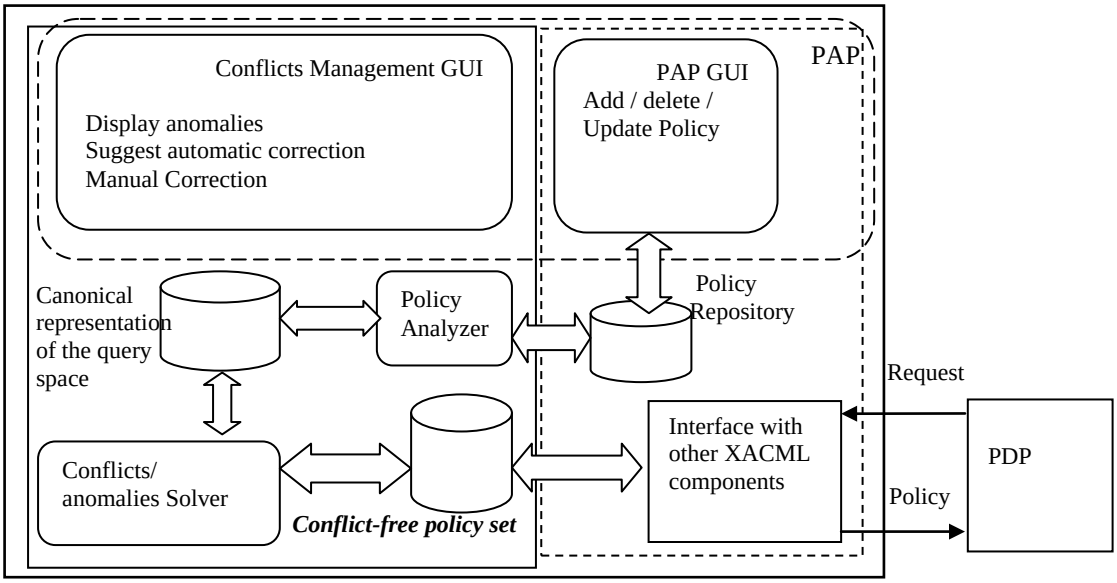

Figure 6. The PAP Architecture After Integration Of The Framework Of Detection And Resolution Of Conflicts/Anomalies

4.1.1 Policy Analyzer

This module is responsible for maintaining up-to-date the canonical representation of the query space. It implements the algorithms for updating the canonical representation following the addition or deletion of policies. It can be invoked, if necessary, in an iterative manner to reconstruct the canonical representation of the query space.

4.1.2. Canonical representation of the query space

This is the collection of the canonical representation elements. For each element $\overline{r^E}$ of the canonical representation the following data are recorded:

- $P(\overline{r^E})$,
- The set of all used attribute names (to simplify algorithms),
- The Boolean expression that define $\overline{r^E}$,
- The resultant policy integration expression in $\overline{r^E}$.

4.1.3 Conflict-free policy set

This is a conflict-free representation of policies. It is constructed by the *Solver* from the canonical representation of the query space. In fact, each element $\overline{r^E}$ of the canonical representation will be translated into a policy such that:

- The policy target is derived from the Boolean expression defining $\overline{r^E}$,
- The policy will be calculated by the Solver in the basis of the resultant policy integration expression in $\overline{r^E}$.

4.1.4. Conflicts/anomalies Solver

This module builds the "*Conflict-free policy set*" from the "*Canonical representation of the query space*" by calculating a resultant policy on each element $\overline{r^E}$. The calculation is based on the following steps:

- Calculate the projection of policies $P(\overline{r^E})$ on $\overline{r^E}$. The aim of this step is to restrict the target for each policy in $\overline{r^E}$,
- Calculate the projection of the policy integration expression on $\overline{r^E}$ that we call in the following "*The resultant policy integration expression*" on $\overline{r^E}$. This can significantly reduce the complexity of the integration expression. Indeed, a policy which is not defined on $\overline{r^E}$ will have P_{NA} as projection. Note that P_{NA} possesses some arithmetic properties which simplify expressions (e.g.: $\forall$ P, P_{NA} & P = P_{NA} et P_{NA} + P =P)

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

- Evaluate the "*resultant policy integration expression*" in $\overline{r}^{E}$ to find the final policy.

Remark We chose to save the policy projections and integration expressions used by the Solver because these policies and these expressions will help to improve the visibility of administrators and allow them to analyze the interference between policies. Indeed, to resolve anomalies, administrators can experience scenarios of the following actions:

(a) Modify an initial policy;

(b) Act directly on a projected policy;

(c) Modify the initial integration expression;

(d) Alter the resultant integration expression.

4.1.5. Conflict Management GUI

The graphical interface for managing conflicts/anomalies will allow administrators to view the authorization classes with card($P(\overline{r}^{E})$) > 1. These are authorization classes that correspond to conflicts/anomalies in the access control system. All information which could assist in the analysis of conflicts should be displayed:

- The list of policies that generate the authorization class,
- The resultant policy integration expression in $\overline{r}^{E}$,
- the Boolean expression defining $\overline{r}^{E}$,
- …

4.2. Algorithms

In this section we implement the results of section III by detailing the most important algorithms necessary for the Framework.

4.2.1. Policy addition

This algorithm will be invoked after the addition of a policy. Its role is to update the canonical representation. It is based on the following results:

Algorithm 1 updating the canonical representation following the addition of a policy

```
/* structure of an element of the canonical representation RΣ */
    listAttrib     // list of used attributes
    listPolicies   // list of policies
    exprBool   // the Boolean expression defining the element

/* algorithm start */
Input:  P      //policy
        RC     // initial canonical representation of RΣ
Output: RC     // the new canonical representation.
Variable : as,          \\ Boolean expression
        listAttrib,     \\ list of attributes
        tempRC     \\ temp canonical representation of RΣ
        rc1        \\ element of the canonical representation

tempRC ← RC
as ← expressionBoolen(P);

listAttrib ← listOfUsedAttrib(P)

/* removal of elements that are not impacted by the addition of P */
foreach attrib ∈ listAttrib do
    foreach rc ∈ tempRC do
        if domainValue((rc.exprBool & as) , attrib) = ϕ
        then tempRC.remove(rc)
        Else RC.remove(rc)
        End if
    End foreach
End foreach

/*  the remaining elements in tempRC will either split into two */
/* or  just add the policy P in their definition*/

foreach rc ∈ tempRC do
    test = false
    foreach attrib ∈ listAttrib do
    /* the domain values of as  is not a subset of rc  rc*/
        if not(domainValue(rc, attrib) ⊂
                domainValue(as, attrib))
        Then test =true
            Break
        End if
    End foreach

 /* the element will be divided in two */
    If test then
        /* creation of the second  element */
        rc1.listeAttrib = rc.listAttrib
        rc1.listPolicies = rc.listPolicies
        rc1.exprBool = rc.exprBool ∧ (¬ as)
        RC.add(rc1)
        /* modification of the first element */
        rc.listAttrib.add (listeAttrib)
        rc.listPolicies.add(P)
        rc.exprBool= rc.exprBool ∧ ¬ as
        RC.add(rc)
  /* the element will just modified by adding P to the policies */
    Else
        rc.listPolicies.addPolicy(p)
        RC.add(rc)
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
    End if
End foreach

return RC
```

4.2.2. Deleting a policy

Algorithm 2 : updating the canonical representation following the deletion of a policy

```
Input:    P     // policy
          RC  //canonical representation
Variable : poliSetTemp  //set of policies,
           asSetTemp  //authorization space

foreach rc ∈ RC  do
    If  (p ∉ rc.getAppPolicies()) then
        asSetTemp.append(rc)
        rc.remove(rc)
    End if
End foreach

foreach rsTemp ∈ asSetTemp  do
    foreach rc ∈ RC  do
        If  (rsTemp.getAppPolicies()\{P}
            = rc.getAppPolicies())
        then
            rc.exprBool = rc.exprBool ∨
                          rsTemp.exprBool
            Break
        End if
    End foreach
  rsTemp.setAppPolicies(rsTemp.setAppPolicies()\{P})
  RC.append(rsTemp)
End foreach

Return RC
```

4.2.3. Calculation of the policy integration expression on canonical representation elements

To evaluate the resultant policy integration expression in an element of the canonical representation, we propose an algorithm based on binary expression trees. It will comprise two steps:

- Step 1: This step can be performed by one of the classical algorithms of construction of binary expression trees from an arithmetic expression (so we skip it here). Then it will have as input the FIA policy integration expression and the result will be the binary expression tree.
- Step 2: the input of this step is the binary tree constructed in Step 1. The algorithm substitutes policies by their projections in the binary tree and then reduces this tree by exploiting the arithmetic properties of the policy P_{NA} (such as: $\forall$ P, P_{NA} & P = P_{NA}, P + P_{NA} =P …). The output of this step is the resultant policy integration expression.

Algorithm 3 Step 2

```
/* node structure */
        char Operator;
        policy myPolicy;
        Node fg, fd;

/* algorithm start */
Input:   arbInteg  // The binary tree of the integration
                   // expression,
         Rc     // canonical representation element.
Output:  expR   // The resultant policy integration
                // expression on  rc.

Variable : expRg, expRd  // policy integration expression

Express_Proj (arbInteg)

/* the node is a leaf then myPolicy is not null */
if arbInteg.Operator = 'c'
Then   expR := myPolicy
/* non-leaf node then we make a recursive call */
Else
    expRg := Express_Proj(arbInteg.fg)
    expRd := Express_Proj (arbInteg.fd)
End if

If arbInteg.Operator = '+' then
    If expRg = PNA then
        If expRd = PNA then
            expR := PNA
        Else expR := expRd
        End if
    Else
        If expRd = PNA then
            expR := expRg
        Else expR := expRg + expRd
        End if
    End if
Else
    if arbInteg.Operator := '&' then
        If expRg = PNA then
            expR := PNA
        Else
            If expRd = PNA then
                expR := PNA
            Else expR := expRg + expRd
            End if
        End if
    End if
End if

Return expR
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

5. CONCLUSION

In this work we proposed a comprehensive Framework for conflicts/anomalies management. We gave special attention to the mathematical formalism of the problem. The canonical representation of the query space proposed in this paper is used to construct an anomaly-free set of policies derived from the initial policy set. This component constitutes the core of the Framework. We also showed the validity of this Framework and the feasibility of implementation by detailing the most important algorithms. The user orientation is one of the strengths of this framework. Indeed, the proposed architecture promotes interactions between the access control system and its administrators.

REFERENCES

[1] P. Rao, D. Lin, E.Bertino, N. Li, and J. Lobo, "An algebra for fine-grained integration of XACML policies," SACMAT 09: Proceedings of the 14th ACM symposium on Access control models and technologies, pp.63-72, 2 New York, NY, USA, ACM, 2009.

[2] H. Hu, G.-J. Ahn, and K. Kulkarni, "Anomaly Discovery and Resolution in Web Access Control Policies," Proceedings of the 16th ACM Symposium on Access Control Models and Technologies, pp.165-174, SACMAT, 2011.

[3] H. Hu, G.-J Ahn and K. Kulkarni, "Ontology-based Policy Anomaly Management for Autonomic Computing," Proceedings of 7th International Conference on Collaborative Computing, Orlando, Florida, USA, pp.487-494, October 15-18, 2011.

[4] E. Bertino, B. Catania, E. Ferrari and P. Perlasca, "A Logical Framework for Reasoning about Access Control Models," ACM Transactions on Information and System Security, vol. 6, No. 1, February 2003, pp.71–127.

[5] G. Hughes and T. Bultan, "Automated verification of access control policies," Technical Report 2004-22, Department of Computer Science, University of California, Santa Barbara, Sept. 2004.

[6] OASIS, "Extensible access control markup language (xacml)" version 2.0. OASIS Standard, 2005.

[7] P. Bonatti, S. D. C. D. Vimercati, and P. Samarati, "An algebra for composing access control policies," ACM Transactions on Information and System Security (TISS), vol. 5, No. 1, February 2002, pp. 1–35.

[8] A. Anderson, "Evaluating xacml as a policy language," Technical report, OASIS, 2003.

[9] M. Abedin, S. Nessa, L. Khan, and B. M. Thuraisingham, "Detection and resolution of anomalies in firewall policy rules", 20th Annual IFIP WG 11.3, Working Conference on Data and Applications Security (DBSec), 2006.

[10] Q. Ni, E. Bertino, J. Lobo, "D-algebra for composing access control policy decisions", Proceedings of ACM Symposium on Information, Computer and Communication Symposium (ASIACCS 2009), 10-12 March.