Expert Systems with Applications 42 (2015) 165–178

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

SBA-XACML: Set-based approach providing efficient policy decision process for accessing Web services

Azzam Mourad *, Hussein Jebbaoui

Department of Computer Science and Mathematics, Lebanese American University (LAU), Lebanon

ARTICLE INFO

Article history:
Available online 31 July 2014

Keywords:
Web services
Security
Set-based algebra
Policy evaluation
Real-time decision
Access control
XACML

ABSTRACT

Policy-based computing is taking an increasing role in providing real-time decisions and governing the systematic interaction among distributed Web services. XACML (eXtensible Access Control Markup Language) has been known as the de facto standard widely used by many vendors for specifying access and context-aware policies. Accordingly, the size and complexity of XACML policies are significantly growing to cope with the evolution of web-based applications and services. This growth raised many concerns related to the efficiency of real-time decision process (i.e. policy evaluation) and the correctness of complex policies. This paper is addressing these concerns through the elaboration of SBA-XACML, a novel Set-Based Algebra (i.e. SBA) scheme that provides efficient evaluation of XACML policies. Our approach constitutes of elaborating (1) a set-based language that covers all the XACML components and establish an intermediate layer to which policies are automatically converted, and (2) a semantics-based policy evaluation that provides better performance compared to the industrial standard Sun Policy Decision Point (PDP) and its corresponding ameliorations. Experiments have been conducted on real-life and synthetic XACML policies in order to demonstrate the efficiency, relevance and scalability of our proposition. The experimental results explore that SBA-XACML evaluation of large and small sizes policies offers better performance than the current approaches, by a factor ranging between 2.4 and 15 times faster depending on policy size.

1. Introduction

Web and cloud services are becoming very popular and constituting the primary techniques for data exchange between distributed systems and partners. Nowadays, several services are being composed (Karakoc & Senkul, 2009; Mourad, Ayoubi, Yahyaoui, & Otrok, 2012) or grouped together into communities (Khosrowshahi Asl, Bentahar, Mizouni, Khosravifar, & Otrok, 2014) in order to form complex systems and provide advanced set of features over the web. However, researchers are still facing the risk of exploits due to the vast accessibility of these services over the Internet (Bhalla & Kazerooni, 2007; Wang, Wang, Xu, Kit Wan, & Vogel, 2004). Moreover, critical services are emerging such as banking and other business transactions, which raise many security challenges. In this regard, policy-based computing (Ayoubi, Mourad, Otrok, & Shahin, 2013; Tout, Mourad, & Otrok, 2013; Yahyaoui, Mourad, AlMulla, Yao, & Sheng, 2012) is taking an increasing role in governing the systematic interaction and composition among distributed services. Particularly, access control is the most challenging aspect of Web service security to determine which partner can access which service. Currently, an increasing trend is to declare policies in a standardized specification language such as XACML, the OASIS standard eXtensible Access Control Markup Language (Moses, 2011). XACML has been known as the de facto standard widely used by many vendors for specifying access control and context-aware policies. It has been emerged as alternative solution to the traditional way of embedding policy verification as part of the application features.

XACML is an XML-based standard for communicating and enforcing access control policies between services and servers (Ayoubi et al., 2013; Moses, 2011). The XACML based policy has complex structure partitioned into three layers: The top layer contains policy sets, the middle layer contains policies and the lower layer contains rules. Each of the three layers has its own target, which contains a set of subjects, resources and actions. Every policy set has a combining algorithm to make the final decision in case of a tie between its policies, and every policy has a combining algorithm to make the final decision in case of a tie between its rules. According to the current XACML engine (Moses, 2011), each request is submitted to the Policy Enforcement Point (PEP) that

* Corresponding author.
E-mail addresses: azzam.mourad@lau.edu.lb (A. Mourad), hussein.jebbaoui@lau.edu.lb (H. Jebbaoui).

http://dx.doi.org/10.1016/j.eswa.2014.07.031
0957-4174/

formulates it using XACML language. Consequently, the Policy Decision Point (PDP) checks at runtime the request with respect to the policy in order to determine access or deny decision. The final decision is enforced by the PEP. This whole process is referred to by policy evaluation. Please refer to Section 7 Listings 1 and 2 for a complete example of a Bank service XACML policy and request evaluation.

With the increase of adopting single, composed and grouped Web services into web-based solutions (Karakoc & Senkul, 2009; Khosrowshahi Asl et al., 2014; Mourad et al., 2012), the size and complexity of XACML policies are significantly growing to cope with this evolution and cover the variety of access conditions. Some real-life composed and grouped policies may nowadays embed hundreds and even thousands of rules. On the other hand, this growth raises many concerns related to the efficiency of real-time decision process of complex policies and makes them candidate for insertion of possible flaws between policies and rules. To elaborate more, XACML evaluation engine is responsible of verifying all the rules of all the participating policies, in addition to resolving their corresponding combining algorithms, in order to handle the decisions to the requests at runtime. Hence, enforcing large size XACML policies will decrease the efficiency of policy evaluation engine, and consequently may create performance bottleneck for the services. Several approaches (Liu, Chen, Hwang, & Xie, 2008; Marouf, Shehab, Squicciarini, & Sundareswaran, 2011; Ngo, Makkes, Demchenko, & de Laat, 2013; Pina Ros, Lischka, & Gómez Mármol, 2012) have been proposed to ameliorate the performance of policy evaluation process of the original XACML engine (Moses, 2011). However, these propositions entail major modification on the Sun PDP architecture (Moses, 2011) and assumptions in terms of continuous policy loading and cumulative reception of all requests, which do not always hold in real world environment and limit their efficiency and usefulness. More details about these limitations are presented in Section 2. Hence, decreasing the overhead of XACML evaluation process still constitutes a real challenge.

In this paper, we address the aforementioned accuracy and performance problems by elaborating a novel set-based approach for the evaluation of XACML policies. The formal specification of policies and rules using sets is allowing us to efficiently perform evaluation and analysis tasks. The proposed SBA-XACML scheme is composed of a formal algebra language including an automatic converter and compiler, and a policy evaluation module based on formal semantics. All the approach components have been implemented in one development framework that accepts XACML policies and requests as inputs, converts them automatically to SBA-XACML constructs when needed and evaluates the requests and policies to provide the final access decision. To download and get additional information about the developed framework and experiments, please visit the following link: http://www.azzammourad.org/#projects. In this context, the main contributions of SBA-XACML are three folds:

- Set-based intermediate representation of XACML constructs into readable mathematical syntax that maintains the same XACML policy structure and accounts for all its elements and their sub elements including rule conditions, obligations, request and response. The corresponding language and compiler offer automatic and optional conversion from XACML to SBA-XACML constructs.
- Formal semantics and algorithms that take advantage of the mathematical operations to provide efficient policy evaluation. Unlike current literature, the adopted approach maintains the same architecture of the industrial standard XACML Sun PDP (Moses, 2011) and respects the major properties and assumptions of real-life environments in terms of remote policy loading upon need and disjoint reception of requests from distributed parties. The experimental results conducted on real-life and synthetic XACML policies explore that SBA-XACML evaluation of large and small size policies provide better performance than Sun PDP (Moses, 2011) and its corresponding ameliorations in the literature (Liu et al., 2008; Marouf et al., 2011; Ngo et al., 2013; Pina Ros et al., 2012), by a factor ranging between 2.4 and 15 times faster depending on policy size.

The rest of the paper is organized as follows. The related work is summarized in Section 2. Section 3 is devoted for the approach overview and architecture. The description of the proposed SBA-XACML language is presented in Section 4. The formal semantics of SBA-XACML policy evaluation is offered in Section 5. In Section 6, the evaluation algorithms are presented. A case study of policy evaluation and a discussion of the experimental results are illustrated in Sections 7 and 8. Finally, the conclusion is presented in Section 9.

2. Related work

In this section, we provide an overview of the related work in the literature addressing XACML policy evaluation and formalization. Few approaches have been proposed in this regards. Liu et al. (2008) proposed the XEngine which is a scheme for efficient XACML policy evaluation. It is an extension to the SUN Policy Decision Point (PDP) (Moses, 2011). Their approach improves the performance of the PDP by numericalization and normalization of the XACML Policies. It consists of 3 steps. The first one is the conversion process of all the strings of XACML based policy and requests to numerical values. The second one is the normalization process which is the conversion of the output from the first step to hierarchical structure and conversion of combining algorithm to First-applicable. The third step is creating a tree structure from the second one. Their approach provides amelioration with respect to policy evaluation performance. However, they do not support obligations due to the conversion of all combining algorithms to first applicable (Ngo et al., 2013). Moreover, the major modification on the Sun PDP architecture and main assumptions of their experiments do not always hold in real world environment, which limit the efficiency and usefulness of their proposition. First, assuming that the policies are always loaded in the memory contradicts with the core concept of XACML (Moses, 2011) and is problematic for large size policies with hundreds and thousands of rules. The policies should be loaded upon request for a short period, where the policy repository can be accessed locally or remotely for security, privacy and memory restriction purposes. Second, our experiments with their tools show that the main overhead reduction is achieved when all the requests (i.e. up to 100,000 requests) are received, converted and loaded in the memory at the same time, then all of them evaluated against the already loaded policies. Again, such assumption does not always hold since requests can be received from different parties at variant time–space. In this regard, the provided experimental results explore that our approach provides better performance than XEngine.

Marouf et al. (2011) proposed a clustering and re-ordering techniques for optimizing XACML performance. The proposed clustering method groups policies and rules based on target subjects. The re-ordering process is based on statistical analysis of policy and vibrant stream of requests. This process reduces the evaluation time because applicable policies and rules are given higher priority to be evaluated first. Although this approach seems interesting, the assumption of access requests following a consistent distribution and policy re-ordering does not support

obligations. Moreover, they share the same limitations as XEngine (Liu et al., 2008) in terms of major modification to Sun PDP architecture and experiments assumptions. The provided results show that our approach offers better performance based on their experiments in Marouf et al. (2011).

Ngo et al. (2013) proposed Multi-Data-Types Interval Decision Diagrams for XACML Evaluation Engine. Their approach is based on data interval partition aggregation along with new decision diagram combinations. They claim that their proposed approach does not only improve the evaluation response time, but also provides correctness and completeness of evaluation. Their proposed approach seems interesting, however it is only experimented on small scale policies up to 360 rules, unlike our and other approaches (Liu et al., 2008; Marouf et al., 2011).

Pina Ros et al. (2012) proposed an optimization for XACML policies evaluation based on two trees. The first tree is a matching tree which is created for a quick finding of applicable rules. The second tree is a combining tree which is used for evaluation of the applicable rules. They proposed a binary search algorithm for finding the matching tree. This approach supports requests with multi-valued attributes, however the matching tree does not support policies with multi-valued attributes.

Rao, Lin, Li, and Lobo (2009) introduced an algebra for fine-grained integration that supports specification of a large variety of integration constraints. They introduced a notion of completeness and prove that their algebra is complete with respect to this notion. Then, they proposed a framework that uses the algebra of fine-grained integration of policies expressed in XACML. Their approach, however, does not cover rule conditions and obligations and focuses on integration between different parties, unlike ours which focuses on analyzing policy sets individually and after integration.

Kolovski, Hendler, and Parsia (2007) proposed a formalization of XACML using description logics (DL), which are a decidable fragment of First-Order logic. They perform policy verification by using the existing DL verifiers. Their analysis service can discover redundancies at the rule level. This approach may also speed up the evaluation process by removing rules that do not affect the final decision. However, they do not support multi-subject requests, complex attribute functions, rule Conditions and Only-One-Applicable combining algorithm.

Mazzoleni, Bertino, and Crispo (2006) proposed an authorization technique for distributed systems with policies from different parties. Their approach is based first on finding similarities between policies from different parties based on requests. Then, it provides an XACML extension by which a party can provide the integration preferences. This approach focuses on policy integration from different parties and do not address policy analysis.

Based on the study of the current literature, it is trivial that this domain is still and will continue to be a challenging niche for researchers. Our approach differs from the aforementioned ones in different aspects. First, it is providing a set-based algebra syntax and semantics that accounts for all the XACML elements, including rules conditions and obligations. Second, it is maintaining the same policy structure of XACML and architecture of Sun PDP, where policies are converted into intermediate mathematical and readable syntax. This allowed us to benefit from the formal description for efficient policy evaluation and analysis purposes, which are not yet addressed by the current propositions. Third, unlike all the current approaches, our scheme is respecting the major properties and assumptions made by Sun PDP (Moses, 2011) with respect to real-life environment, where policies are loaded from local or remote location upon need, and the XACML requests are received one at a time from distributed parties. Finally, our experiments in Section 7 show that our proposition outperforms the current approaches.

3. Approach overview and architecture

In this section, we present the overall architecture of our approach illustrated in Fig. 1.

Our proposition includes the SBA-XACML Language, Compiler and Evaluation Module. All the approach components have been implemented in one development framework that accepts XACML policies and requests as inputs, convert them to SBA-XACML when needed, and perform systematically and automatically all the evaluation processes. It may also accepts already converted or written SBA-XACML policies and requests. Using the framework, the user can evaluate the requests and get the corresponding responses using the module embedding the evaluation algorithms. To download and get additional information about the developed framework and experiments, please visit the following link: http://www.azzammourad.org/#projects.

3.1. SBA-XACML language & compiler

SBA-XACML is a formal language based on algebra sets and composed of all the elements and constructs needed for the specification of XACML based policy, request and response. Section 4 presents the complete definition and syntax of SBA-XACML elements and attributes. SBA-XACML compiler includes XACML parser and converter to SBA-XACML. It takes XACML policy set and request as inputs, parses their XACML elements and generates SBA-XACML constructs according to the language syntax and structure. It can be used independently to convert XACML to SBA-XACML, or as embedded in our framework with the policy evaluation module to convert XACML to SBA-XACML at run time.

3.2. Policy evaluation module

This module allows to evaluate a SBA-XACML request against a set of SBA-XACML policies. It is composed of policy-level and rule-level evaluation algorithms (see Section 6) that realize the

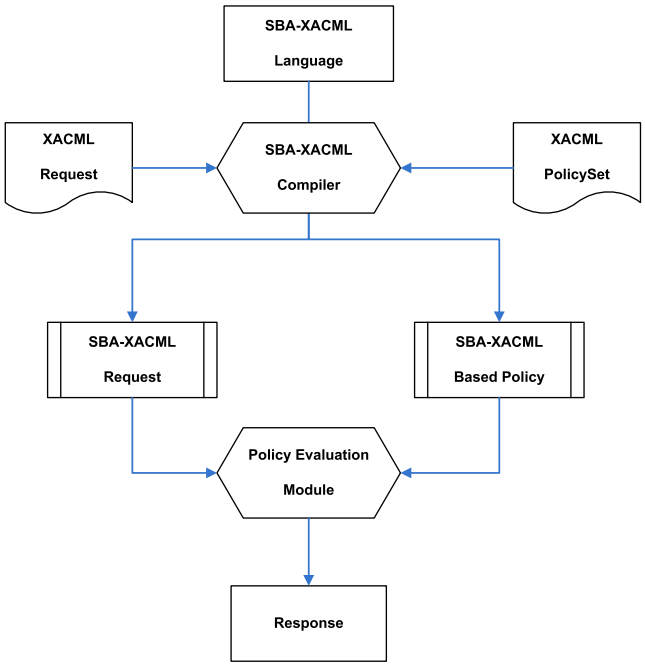


Fig. 1. SBA-XACML architecture.

elaborated policy evaluation semantics presented in Section 5. The policy-level algorithm is responsible for evaluating the policies and triggers the rule-level one in order to evaluate the rules in each policy. It accepts as inputs a SBA-XACML request and a policy set.

4. SBA-XACML language description

A set-based algebra is an accumulation of distinct mathematical elements that describes the fundamental characteristics and includes the regulations of sets and other operations such as union, intersections, complementation, equality and inclusion. It additionally provides systematic procedures for evaluating expressions and performing calculations, involving these operations and relations. SBA-XACML is a set-based algebra language. In the following, we present its constructs, operators and structure: ($PS; P\ (R; Rq)$; Rs (“Permit”; “Deny”; “NotApplicable”; “Indeterminate”); op ($\subseteq; \subset; \wedge; \rangle; \vee; \cap$)), where.

- PS: represents a policy set (or a based policy) which is composed of one or more policies.
  - P: represents a policy which is composed of one or more rules.
  - R: represents a rule.
- Rq: represents a request.
- Rs: represents a response that contains the final decision.
  - “Permit”, “Deny”, “NotApplicable” and “Indeterminate” are policy constants and represent the final decision embedded in the response.
- op: represents an operator.
  - $\subseteq$: represents a subset or equal.
  - $\subset$: represents a subset.
  - $\wedge$: represents logical operator ”and”.
  - $\vee$: represents logical operator ”or”.
  - -> : represents the precedence order between operations.
  - $\cap$: represents the intersection between two sets.

In the sequel, we present the SBA-XACML syntax for the based policy, request and response respectively.

4.1. SBA-XACML based policy

XACML based policy, which they also refer to as a policy set PS, is ordered into 3 levels: $PolicySet, Policy$, and $Rule$. Every element can contain a $Target$. $PolicySet$ element contains other $PolicySet(s)$ and/or $Policie(s)$. $Policy$ contains $Rule(s)$. $PolicySets$ and $Policies$ have their $Obligations$ to fulfill whenever a $Response$ is reached to either a $Permit$ or $Deny$ decision. In the following, we present the definitions and syntax of all the elements.

4.1.1. Common elements definitions and syntax

The following are the common elements that are used at the policy set, policy and rule levels.

A target TR is an objective and is mapped to SBA-XACML within the context of rule, policy and policy set according to the following syntax:

$$TR = \{S, R, A\} \qquad (\textbf{Construct 1})$$

where S is a set of subjects, R is a set of resources and A is a set of actions.

Obligations $OBLs$ contain one or more obligation(s) OBL. An obligation is an action that takes place after a decision has been reached to either $Permit$ or $Deny$. It is mapped to SBA-XACML within the context of policy and policy set according to the following syntax:

$$OBLs = OBL - Set \qquad (\textbf{Construct 2})$$
$$OBL = \{OBLID, FFOn, \{\{AttID, DT, V\}\}\} \qquad (\textbf{Construct 3})$$

where $OBL - Set$ is the the set of obligation OBL to be performed, $OBLID$ is the id identifying the obligation, $FFOn$ is the Fulfill On attribute that is used as a key to determine when the obligation must be enforced and must be either permit or deny, $AttID$ is the attribute id of the obligation to be carried out, DT is the data type and V is the value. If the policy or policy set being evaluated matches the $FFOn$ attribute of its obligations, then the obligations are passed to be enforced otherwise obligations are ignored.

4.1.2. PolicySet (PS) definition and syntax

A policy set PS is a container of policies. PS may contain other policy sets, policies or both. It can also be referenced by other policy sets. It is mapped to SBA-XACML according to the following syntax:

$$PS ::= \langle ID, SP, PR, PCA, IPS, OBLs, TR\rangle \qquad (\textbf{Construct 4})$$

where ID is the policy set id, SP is the set of policies that belongs to policy set PS, PR is the precedence order of policies that belongs to PS, PCA is the policy combining algorithm, IPS is the policies or policy set that are referenced by $PS, OBLs$ is the set of obligations and TR is the target (refer to Section 4.1.1 for details).

Example 1. Consider a policy set $PS1$ with two policies $P1$ and $P2$. $PS1$ has a $PCA = deny - overrides$. $PS1$ has a target $subject = Bob, resource = FileA$ and $action = Read$. It has no reference to other policies and no obligations. The policy set $PS1$ is mapped to SBA-XACML as follows:

$$PS :: = \langle PS1, \{P1, P2\}, \{P1 > P2\},$$
$$\{deny - overrides\}, \{\}, \{\}, \{\{Bob\}, \{FileA\}, \{Read\}\}\rangle$$

4.1.3. Policy (P) definition and syntax

A policy P is a single access control policy. It is expressed through a set of rules. A policy contains a set of rules, rule combining algorithm, target and obligations. It is mapped to SBA-XACML according to the following syntax:

$$P ::= \langle ID, SR, PR, RCA, OBLs, TR\rangle \qquad (\textbf{Construct 5})$$

where ID is the policy id, SR is the set of rules that belongs to policy P, PR is the precedence order of rules that belongs to P, RCA is the rule combining algorithm, $OBLs$ is the set of obligations and TR is the target (refer to Section 4.1.1 for details).

Example 2. Consider a policy $P1$ with two rules $R1$ and $R2$. $P1$ has a $rulecombiningalgorithm = permit - overrides$. It has a target $subject = Bob, resource = FileA$ and $action = write$ and without any obligations. The policy $P1$ is mapped to SBA-XACML as follows:

$$P :: = \langle P1, \{R1, R2\}, \{R1 > R2\}, \{permit - overrides\}, \{\},$$
$$\{\{Bob\}, \{FileA\}, \{write\}\}\rangle$$

4.1.4. Rule (R) definition and syntax

A rule R is the most elementary element of a policy. A rule contains rule conditions, target and rule effect. It is mapped to SBA-XACML according to the following syntax:

$$R ::= \langle ID, RC, TR, RE\rangle \qquad (\textbf{Construct 6})$$

where ID is the rule id, RC is the set of rule conditions, TR is the target (refer to Section 4.1.1 section for details), and RE is the rule effect.

A rule condition *RC* is a boolean function over subjects, resources, actions or functions of attributes. It is mapped to SBA-XACML within the context of a rule according to the following syntax:

$$RC = \{Apply_function, \{parameters\}\} \quad (\textbf{Construct 7})$$

where $Apply_function$ is the function used in evaluating the elements in the apply and *parameters* are the input to the function being applied, each of which is an evaluatable.

Example 3. Consider a rule $R1$ with $ruleeffect = permit$. $R1$ has no target defined. Its only condition is that anyone accessing *File*1 is allowed at any time. The rule is mapped to SBA-XACML as follows:

$$R ::= \langle R1, \{\{string-equal, \{ResourceAttributeDesignator, string, File1\}\}\}, \{\}, \{\}, \{\}\}, \{Permit\}\rangle$$

4.2. A-XACML request

A request *Rq* is a call for access to some resources. It is mapped to SBA-XACML according to the following syntax:

$$Rq ::= \langle Sr, Rr, Ar\rangle \quad (\textbf{Construct 8})$$

where *Sr* is the set of subjects, *Rr* is the set of resources and *Ar* is the set of actions.

Example 4. Consider a request calling for access with subject *Bob*, resource *ServerA* and action *read*. The request is mapped to SBA-XACML as follows:

$$Rq ::= \langle \{Bob\}, \{ServerA\}, \{Read\}\rangle$$

4.3. A-XACML response

A response *Rs* is a decision to a request against a based policy. It is mapped to SBA-XACML according to the following syntax:

$$Rs ::= \langle D, OBLs\rangle \quad (\textbf{Construct 9})$$

where *D* is the decision of the response and *OBLs* is the set of obligations to be executed within the response (refer to Section 4.1.1 section for details).

Example 5. The response to the request in Example 4 is mapped to SBA-XACML as follows:

$$Rs ::= \langle \{permit\}, \{\}\rangle$$

5. Evaluation semantics

Formal semantics constitutes of rigorous mathematical study of the meaning of languages and models of computation (Pagan, 1981; Slonneger & Kurtz, 1995). The formal semantics of a language is specified by a mathematical model that illustrates the possible computations described by the language. Operational semantics describes the execution of the language directly rather than by translation. It somehow corresponds to interpretation, where the implementation language of the interpreter is a mathematical formalism. The structural operational semantics used in this paper is an approach proposed to give logical means in defining operational semantics (Plotkin, 2004). It consists of defining the behavior of a process in terms of the behavior of its parts. Computation is represented by means of deductive systems that turn the abstract machine into a system of logical inferences. This allows to apply formal analysis on the behavior of processes. The proofs of process properties are derived directly from the definitions of the language constructs because the semantics descriptions are based on deductive logic. With structural operational semantics, the behavior of a process is defined in terms of a set of transition relations. Such specifications take the form of inference rules. The valid transitions of a composite piece of syntax is defined into these rules in terms of the transitions of its components. Definitions are given by inference rules, which consist of a conclusion that follows from a set of premises, possibly under control of some conditions. An inference rule has a general form consisting of the premises listed above a horizontal line, the conclusion below, and the condition, if present, to the right (Slonneger & Kurtz, 1995).

In this section, we present the formal semantics of a SBA-XACML policy evaluation following the above inference rule structure and deductive logic. Given a policy set *PS* and a request *Rq*, the response *Rs* is derived by the evaluation $\underset{eval}{\longrightarrow}$ of all premises combined between each other using designated operators *op* as follows:

$$\frac{(premise_1)\ \ op\ \ (premise_2)\ \ op\ \ \ldots\ \ op\ \ (premise_n)}{\langle PS, Rq\rangle \underset{eval}{\longrightarrow} Rs}\ (conclusion)$$

The policy and rule evaluation semantics rules, which constitute the premises in the above rule, have also similar structure and follows the deductive logic until reaching the basic defined premise (i.e. condition). Throughout the rest of the paper, please note the difference between semantics rule that expresses the evaluation at a particular level, and a policy rule which is a construct in SBA-XACML. All the semantics rules follow the bottom up structure, where all the common ones are presented first, then followed by the rule level, policy level and policy set level ones.

5.1. Match function

In this section, we present the matching semantics rules for a request *Rq* with subject set *Sr*, resource set *Rr* and action set *Ar* against a target *TR* with subject set *S*, resource set *R* and action set *A*. The semantics of matching a request and a target is determined by comparing the request subject set *Sr* with target subject set *S*, request resource set *Rr* with target resource set *R* and request action set *Ar* with target action set *A*.

Rules *1* and *2* in Table 1 describe the different matching cases for a request *Rq* with a target *TR*. In Rule *1*, a target *TR* matches a request *Rq* if the requested subject set *Sr* intersects with the target subject set *S*, the requested resource set *Rr* intersects with

Table 1
Match function semantics.

$\frac{((Sr \cap S) \neq \emptyset) \wedge ((Rr \cap R) \neq \emptyset) \wedge ((Ar \cap A) \neq \emptyset)}{\langle TR, Rq\rangle \underset{match}{\vdash} True}$	(**Rule 1**)
$\frac{((Sr \cap S) = \emptyset) \vee ((Rr \cap R) = \emptyset) \vee ((Ar \cap A) = \emptyset)}{\langle TR, Rq\rangle \underset{match}{\vdash} False}$	(**Rule 2**)

Table 2
Evaluation semantics of a policy rule.

$\frac{(\langle TR, Rq\rangle \underset{match}{\vdash} True) \wedge (RC = True) \wedge (RE = Permit)}{\langle R, Rq\rangle \underset{eval}{\longrightarrow} Permit}$	(**Rule 3**)
$\frac{(\langle TR, Rq\rangle \underset{match}{\vdash} True) \wedge (RC = True) \wedge (RE = Deny)}{\langle R, Rq\rangle \underset{eval}{\longrightarrow} Deny}$	(**Rule 4**)
$\frac{(\langle TR, Rq\rangle \underset{match}{\vdash} False) \vee (RC = False)}{\langle R, Rq\rangle \underset{eval}{\longrightarrow} NotApplicable}$	(**Rule 5**)

Table 3
Evaluation semantics of a policy where (RCA = Permit-Overrides).

$\frac{(RCA = Permit-Overrides) \wedge (\langle TR,Rq\rangle \underset{match}{\vdash} True) \wedge (\exists R \in SR; \langle R,Rq\rangle \underset{eval}{\longrightarrow} Permit)}{\langle P,Rq\rangle \underset{eval}{\longrightarrow} Permit, OBLs}$	(**Rule 6**)
$\frac{(RCA = Permit-Overrides) \wedge (\langle TR,Rq\rangle \underset{match}{\vdash} True) \wedge (\forall R \in SR; \langle R,Rq\rangle \underset{eval}{\longrightarrow} Deny)}{\langle P,Rq\rangle \underset{eval}{\longrightarrow} Deny, OBLs}$	(**Rule 7**)
$\frac{(RCA = Permit-Overrides) \wedge ((\langle TR,Rq\rangle \underset{match}{\vdash} False) \vee (\forall R \in SR; \langle R,Rq\rangle \underset{eval}{\longrightarrow} NotApplicable))}{\langle P,Rq\rangle \underset{eval}{\longrightarrow} NotApplicable}$	(**Rule 8**)

Table 4
Evaluation semantics of a policyset where (PCA = Permit-Overrides).

$\frac{(PCA = Permit-Overrides) \wedge (\langle TR,Rq\rangle \underset{match}{\vdash} True) \wedge (\exists P \in SP; \langle P,Rq\rangle \underset{eval}{\longrightarrow} Permit)}{\langle PS,Rq\rangle \underset{eval}{\longrightarrow} Permit, OBLs}$	(**Rule 9**)
$\frac{(PCA = Permit-Overrides) \wedge (\langle TR,Rq\rangle \underset{match}{\vdash} True) \wedge (\forall P \in SP; \langle P,Rq\rangle \underset{eval}{\longrightarrow} Deny)}{\langle PS,Rq\rangle \underset{eval}{\longrightarrow} Deny, OBLs}$	(**Rule 10**)
$\frac{(PCA = Permit-Overrides) \wedge ((\langle TR,Rq\rangle \underset{match}{\vdash} False) \vee (\forall P \in SP; \langle P,Rq\rangle \underset{eval}{\longrightarrow} NotApplicable))}{\langle PS,Rq\rangle \underset{eval}{\longrightarrow} NotApplicable}$	(**Rule 11**)

the target resource set R and the requested action set Ar intersects with the target action set A. In Rule *2*, a target TR does not match a request Rq if the requested subject set Sr does not intersects with the target subject set S, the requested resource set Rr does not intersect with the target resource set R or the requested action set Ar does not intersect with the target action set A.

5.2. Rule evaluation

In this section, we present the evaluation semantics for a request Rq at the policy rule level.

Semantics Rules *3*, *4* and *5* in Table 2 describe the different evaluation cases for a policy rule R. In semantics Rule *3*, a policy rule R evaluates a request Rq to *Permit* if the target matches with the request elements (see details in semantics Rule *1*) and rule conditions RC evaluate to *True* and rule effect RE is *Permit*. In semantics Rule *4*, a policy rule R evaluates a request Rq to *Deny* if the target matches with the request elements and rule conditions RC evaluate to *True* and rule effect RE is *Deny*. In semantics Rule *5*, a policy rule R evaluates a request Rq to *NotApplicable* if either the target does not match with the request elements (see details in semantics Rule *2*) or rule conditions RC evaluate to *False*.

5.3. Policy evaluation

In this section, we present the evaluation semantics for a request Rq at the policy level. Rules *6*, *7* and *8* in Table 3 describe the cases where the rule combining algorithm (RCA) is $Permit-Overrides$. We also elaborated in similar way semantics rules that cover the cases where (RCA = *Deny-Overrides*) and (RCA = *First-Applicable*). However, we do not present them due to space limitation. Algorithm 2 provides some details about them.

In Rule *6*, a policy P evaluates a request Rq to *Permit* with a list of obligations $OBLs$ if the target matches with the request elements (see details in semantics Rule *1*) and there exists a rule R in the set of Rules SR that evaluates to *Permit* (see details in semantics Rule *3*). In Rule *7*, a policy P evaluates a request Rq to *Deny* with a list of obligations $OBLs$ if the target matches with the request elements and all rules in the set of Rules SR that evaluates to *Deny* (see details in semantics Rule *4*). In Rule *8*, a policy P evaluates a request Rq to *NotApplicable* if either the target does not match with the request elements (see details in semantics Rule *2*) or all rules in the set of Rules SR that evaluates to *NotApplicable* (see details in semantics Rule *5*).

5.4. PolicySet evaluation

In this section, we present the evaluation semantics for a request Rq at the policy set level. Rules *9*, *10* and *11* in Table 4 describe the cases where the policy combining algorithm (PCA) is $Permit-Overrides$. We also elaborated in similar way semantics rules that cover the cases where (PCA = *Deny-Overrides*), (PCA = *First-Applicable*) and (PCA = *Only-One-Applicable*). However, we do not provide them due to space limitation. Algorithm 3 provides some details about them.

In Rule *9*, a policy set PS evaluates a request Rq to *Permit* with a list of obligations $OBLs$ if the target matches with the request elements (see details in semantics Rule *1*) and there exists a policy P in the set of policies SP that evaluates to *Permit* (see details in semantics Rules *6*). In Rule *10*, a policyset PS evaluates a request Rq to *Deny* with a list of obligations $OBLs$ if the target matches with the request elements and all policies in the set of policies SP that evaluates to *Deny* (see details in semantics Rules *7*). In Rule *11*, a policy set PS evaluates a request Rq to *NotApplicable* if either the target does not match with the request elements (see details in semantics Rule *2*) or all policies in the set of policies SP that evaluates to *NotApplicable* (see details in semantics Rules *8*).

6. Evaluation algorithms

In this section, we present the algorithms realizing the SBA-XACML policy evaluation semantics. We divided the evaluation module into three algorithms. Each one of them evaluates the request at a separate layer in the based policy.

6.1. Rule evaluation algorithm

The rule evaluation algorithm in Algorithm 1 evaluates the request at the lowest level in the policy set. It takes two inputs: a rule R and a request Rq. The output is the rule decision, which is $Deny, Permit$, or $NotApplicable$.

Algorithm 1. Rule_Evaluation(R, Rq)

Input : 1) A Rule R with Target TR = {S,R,A} and (2) A Request Rq = {Sr,Rr,Ar}
Output : Rule decision ∈ {RE, NotApplicable} where RE ∈ Permit, Deny

```
1: if ((Sr ∩ S) ≠ ∅) ∧ ((Rr ∩ R) ≠ ∅) ∧ ((Ar ∩ A) ≠ ∅) then      ▷ Applicability check
2:     for i := 1 to m do
3:         if ∃ i, 0 < i ≤ m, RC = "false" then                  ▷ If one RC evaluates to false
4:             return NotApplicable;
5:         else                                                  ▷ If all RCs evaluate to true
6:             return RE;
7:         end if
8:     end for
9: else
10:    return NotApplicable;                                     ▷ Not applicable target
11: end if
```

Algorithm 1 begins by checking whether the rule is applicable to the request (line 1). The applicability check is done by comparing

the request set of subjects, set of resources and set of actions against the rule target. If the applicability check returns true, then the rule conditions are evaluated (line 3), otherwise "NotApplicable" is returned to the Policy Evaluation in Algorithm 2 (line 10). Rule effect is returned if all rule conditions evaluate to true (line 6), otherwise "NotApplicable" is returned to the Policy Evaluation in Algorithm 2 (line 4).

6.2. Policy evaluation algorithm

The policy evaluation algorithm in Algorithm 2 evaluates the request at the middle layer. It calls the rule evaluation algorithm Algorithm 1 to handle the evaluation at the lower layer. It takes two inputs: a policy *P* and a request *Rq*. The output is the policy decision which is *Deny*, *Permit*, or *NotApplicable*.

Algorithm 2. Policy_Evaluation(P, Rq)

Input : 1) A Policy P with Target TR = {S,R,A} and (2) A Request Rq = {Sr,Rr,Ar}

Output : Policy decision ∈ {Deny,Permit,NotApplicable}

```
1: if ((Sr ∩ S) ≠ ∅) ∧ ((Rr ∩ R) ≠ ∅) ∧ ((Ar ∩ A) ≠ ∅) then      ▷ Applicability check
2:     for i := 1 to m do                                          ▷ Check all rules
3:         REi = RULE_EVALUATION(Ri, Rq);                          ▷ Call Rule Evaluation Algorithm
4:         if (RCA = "Deny-Overrides") then
5:             if ∃ i, 0 < i ≤ m, REi = "Deny" then                ▷ If one rule evaluates to Deny
6:                 return Deny;
7:             else
8:                 if ∀ i, 0 < i ≤ m, REi = "Permit" then          ▷ If all rules evaluate to Permit
9:                     return Permit;
10:                else
11:                    return NotApplicable;
12:                end if
13:            end if
14:        end if
15:        if (RCA = "Permit-Overrides") then
16:            if ∃ i, 0 < i ≤ m, REi = "Permit" then              ▷ If one rule evaluates to Permit
17:                return Permit;
18:            else
19:                if ∀ i, 0 < i ≤ m, REi = "Deny" then            ▷ If all rules evaluate to Deny
20:                    return Deny;
21:                else
22:                    return NotApplicable;
23:                end if
24:            end if
25:        end if
26:        if (RCA = "First-Applicable") then
27:            if ∃ i, 0 < i ≤ m, REi = "Permit" then              ▷ If first rule evaluates to Permit
28:                return Permit;
29:            else
30:                if ∃ i, 0 < i ≤ m, REi = "Deny" then            ▷ If first rule evaluates to Deny
31:                    return Deny;
32:                else
33:                    return NotApplicable;
34:                end if
35:            end if
36:        end if
37:    end for
38: else
39:    return NotApplicable;                                       ▷ Policy is NotApplicable by target
40: end if
```

Algorithm 2 is composed of three steps: the applicability check of the policy, the evaluation of rules and rule combining algorithm *RCA*. The applicability check is done by matching the request subjects, resources and actions with policy target (line 1). If line 1 returns true, we call step 2, otherwise *NotApplicable* is returned to the policy set evaluation in Algorithm 3. The evaluation of all the rules is done by the order they are listed in the policy. The rule evaluation algorithm Algorithm 1 is called on line 3. The response returned is passed to step 3, where *RCA* can have one of the following values: $Permit - Overrides$ (line 15), $Deny - Overrides$ (line 4) or $First - Applicable$ (line26). Step 3 of the evaluation process differs based on the value of *RCA* in order to provide the decision.

6.3. PolicySet evaluation algorithm

The policyset evaluation algorithm in Algorithm 3 calls the policy evaluation algorithm Algorithm 2 to handle the evaluation at the middle layer. The algorithm takes two inputs: a policy set *PS* and a request *Rq*. The output is the final response to the request *Rs*.

Algorithm 3. PolicySet_Evaluation(PS, Rq)

Input : (1) A PolicySet PS with Target TR = {S,R,A} and (2) Request Rq = {Sr,Rr,Ar}

Output : Request response Rs ∈ {Permit, Deny, NotApplicable, Indeterminate}

```
1: if ((Sr ∩ S) ≠ ∅) ∧ ((Rr ∩ R) ≠ ∅) ∧ ((Ar ∩ A) ≠ ∅) then      ▷ Applicability Check
2:     for i := 1 to m do                                          ▷ Check Policies
3:         PEi = POLICY_EVALUATION(Pi, Rq);                        ▷ Call Policy Evaluation Algorithm
4:         if (PCA = "Deny-Overrides") then
5:             if ∃ i, 0 < i ≤ m, PEi = "Deny" then                ▷ If one policy evaluates to Deny
6:                 return Deny;
7:             else
8:                 if ∀ i, 0 < i ≤ m, PEi = "Permit" then          ▷ If all policies eval. to Permit
9:                     return Permit;
10:                else
11:                    return NotApplicable;
12:                end if
13:            end if
14:        end if
15:        if (PCA = "Permit-Overrides") then
16:            if ∃ i, 0 < i ≤ m, PEi = "Permit" then              ▷ If one policy evaluates to Permit
17:                return Permit;
18:            else
19:                if ∀ i, 0 < i ≤ m, PEi = "Deny" then            ▷ If all policies evaluate to Deny
20:                    return Deny;
21:                else
22:                    return NotApplicable;
23:                end if
24:            end if
25:        end if
26:        if (PCA = "First-Applicable") then
27:            if ∃ i, 0 < i ≤ m, PEi = "Deny" then                ▷ If first policy evaluates to Deny
28:                return Deny;
29:            else
30:                if ∃ i, 0 < i ≤ m, PEi = "Permit" then          ▷ If first policy eval. to Permit
31:                    return Permit;
32:                else
33:                    return NotApplicable;
34:                end if
35:            end if
36:        end if
37:        if (PCA = "Only-one-Applicable") then
38:            if ∃! i, 0 < i ≤ m, PEi = "Deny" then               ▷ Iff one policy evaluates to Deny
39:                return Deny;
40:            end if
41:            if ∃! i, 0 < i ≤ m, PEi = "Permit" then             ▷ Iff one policy evaluates Permit
42:                return Permit;
43:            end if
44:            if ∀ i, 0 < i ≤ m, PEi = "NotApplicable" then
45:                return NotApplicable;                           ▷ If all policies evaluate to NotApplicable
46:            else
47:                return Indeterminate;                           ▷ If more than one policy are Applicable
48:            end if
49:        end if
50:    end for
51: else
52:    return NotApplicable;                                       ▷ PolicySet is NotApplicable by Target
53: end if
```

Algorithm 3 is composed of three steps: the applicability check of the policy set, evaluation of policies and policy combining algorithm *PCA*. The applicability check is done by evaluating the request subjects, resources and actions with the policy set target (line 1). If it returns true, then step 2 is called, otherwise $Rs = NotApplicable$ is returned as the request response. The evaluation of all the policies is done by the order they are listed in the based policy. The policy evaluation algorithm Algorithm 2 is called on line 3. The returned response, which is saved in *PEi*, is passed to step 3, where the *PCA* can have one of the following values: $Permit - Overrides$ (line 15), $Deny - Overrides$ (line 4),

First – Applicable (line 26) or *Only– one – Applicable* (line 37). Step 3 of the evaluation process differs based on the value of *PCA* in order to provide the final decision.

7. Case Study: XACML vs SBA-XACML policy evaluation

In this section, we present a case study illustrating the usability of SBA-XACML policy evaluation process through semantics rules.

7.1. XACML policy evaluation

To start with, we provide an XACML based policy and request, and the response of the evaluation process according to XACML syntax and Sun PDP engine (Moses, 2011). A based policy for a Bank service is presented in Listings 1 and 2. The policy set contains two policies *P*1 and *P*2. The policy set *ID* and policy combining algorithm *PCA* are stated in line 3. *P*1 (lines 4–50) contains two rules *R*1 and *R*2, has a rule combining algorithm *permit – overrides* (line 4) and a policy target (lines 5–29) with subjects equal to *Jerry* and *Bob* (lines 9 and 13), resource *BankService/withdraw* (line 21) and actions *Any* (line 27). *R*1 (lines 30–41) has a *Permit* rule effect thats allows access to *BankService/withdraw* resource if the subject is *Bob*. *R*2 (line 42) denies access to any resources for any subjects. If policy P1 evaluates to *Permit*, it has an obligation (lines 44–50) to send an email to *Customer_service@bank.com*. *P*2 (lines 51–85) con-

```
[1]. <!--Bank Based Policy to Deposit and Withdraw -->
[2]. <?xml version="1.0" encoding="UTF-8"?>
[3]. <PolicySet xmlns="schema:os" PolicyCombiningAlgId="policy-combining-
    algorithm:permit-overrides" PolicySetId="PS1">
[4].   <Policy PolicyId="P1" RuleCombiningAlgId="rule-combining-algorithm:
    permit-overrides">
[5].     <Target>
[6].       <Subjects>
[7].         <Subject>
[8].           <SubjectMatch MatchId="function:string-equal">
[9].             <AttributeValue DataType="string">Jerry</AttributeValue>
[10].              <SubjectAttributeDesignator AttributeId="subject:subject-
    id" DataType="string" />
[11].            </SubjectMatch>
[12].            <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function
    :string-equal">
[13].              <AttributeValue DataType="string">Bob</AttributeValue>
[14].              <SubjectAttributeDesignator AttributeId="subject:subject-
    id" DataType="string" />
[15].            </SubjectMatch>
[16].          </Subject>
[17].       </Subjects>
[18].       <Resources>
[19].         <Resource>
[20].           <ResourceMatch MatchId="function:string-equal">
[21].             <AttributeValue DataType="string">BankService/withdraw</
    AttributeValue>
[22].             <ResourceAttributeDesignator AttributeId="resource:
    resource-id" DataType="string" />
[23].           </ResourceMatch>
[24].         </Resource>
[25].       </Resources>
[26].       <Actions>
[27].         <AnyAction />
[28].       </Actions>
[29].     </Target>
[30].     <Rule Effect="Permit" RuleId="R1">
[31].       <Condition>
[32].         <Apply FunctionId="function:and">
[33].           <Apply FunctionId="function:string-equal">
[34].             <ResourceAttributeDesignator AttributeId="resource:
    resource-id" DataType="string">BankService/withdraw</
    ResourceAttributeDesignator>
[35].           </Apply>
[36].           <Apply FunctionId="function:string-equal">
[37].             <SubjectAttributeDesignator AttributeId="subject:subject-
    id" DataType="string">Bob</SubjectAttributeDesignator>
[38].           </Apply>
[39].         </Apply>
[40].       </Condition>
[41].     </Rule>
[42].     <Rule Effect="Deny" RuleId="R2" />
[43].       <Target/>
```

Listing 1. XACML policy for a bank service part I.

```
[44].     <Obligations>
[45].       <Obligation FulfillOn="Permit" ObligationId="Withdraw">
[46].         <AttributeAssignment AttributeId="example:attribute:mailto"
    DataType="string">Customer_service@bank.com</AttributeAssignment>
[47].         <SubjectAttributeDesignator AttributeId="subject:subject-id"
    DataType="string">subject:subject-id</SubjectAttributeDesignator>
[48].         <ResourceAttributeDesignator AttributeId="resource:resource-id
    " DataType="string">resource:resource-id</ResourceAttributeDesignator>
[49].       </Obligation>
[50].     </Obligations>
[51].   </Policy>
[52].   <Policy PolicyId="P2" RuleCombiningAlgId="rule-combining-algorithm:
    permit-overrides">
[53].     <Target/>
[54].       <Rule Effect="Permit" RuleId="R3">
[55].         <Condition>
[56].           <Apply FunctionId="function:string-equal">
[57].             <ResourceAttributeDesignator AttributeId="resource:
    resource-id" DataType="string">BankService/deposit</
    ResourceAttributeDesignator>
[58].           </Apply>
[59].         </Condition>
[60].       </Rule>
[61].       <Rule Effect="Permit" RuleId="R4">
[62].         <Condition>
[63].           <Apply FunctionId="function:and">
[64].             <Apply FunctionId="function:string-equal">
[65].               <ResourceAttributeDesignator AttributeId="resource:
    resource-id" DataType="string">BankService/deposit</
    ResourceAttributeDesignator>
[66].             </Apply>
[67].             <Apply FunctionId="function:string-equal">
[68].               <SubjectAttributeDesignator AttributeId="subject:subject
    -id" DataType="string">Joe</SubjectAttributeDesignator>
[69].             </Apply>
[70].           </Apply>
[71].         </Condition>
[72].       </Rule>
[73].       <Rule Effect="Deny" RuleId="R5">
[74].         <Condition>
[75].           <Apply FunctionId="function:and">
[76].             <Apply FunctionId="function:string-equal">
[77].               <ResourceAttributeDesignator AttributeId="resource:
    resource-id" DataType="string">BankService/deposit</
    ResourceAttributeDesignator>
[78].             </Apply>
[79].             <Apply FunctionId="function:string-equal">
[80].               <SubjectAttributeDesignator AttributeId="subject:subject
    -id" DataType="string">Joe</SubjectAttributeDesignator>
[81].             </Apply>
[82].           </Apply>
[83].         </Condition>
[84].       </Rule>
[85].   </Policy>
[86]. </PolicySet>
```

Listing 2. XACML policy for a bank service part II.

tains three rules $R3, R4$ and $R5$, has a rule combining algorithm *permit − overrides* (line 52) and no target. $R3$ (lines 54–60) has a *Permit* rule effect that allows access to *BankService/deposit* resource for any subjects. $R4$ (lines 61–72) has a *Permit* rule effect that allows access to *BankService/deposit* resource if the subject is *Joe*. $R5$ (lines 73–84) has a *Deny* rule effect that prohibit access to *BankService/deposit* resource if the subject is *Joe*.

Listing 3 contains the XACML request. The request is calling for a resource *BankService/deposit* with a subject *Bob* and action *execute*. Lines 4,9 and 14 contain subject, resource and action respectively.

Listing 4 contains the XACML response to the request in Listing 3 against the based policy in Listings 1 and 2. The response *Permit* is the final decision for the resource *BankService/deposit* with a subject *Bob* and action *execute*.

7.2. SBA-XACML policy evaluation

In the sequel, we provide in Listing 5 the generated SBA-XACML based policy corresponding to the XACML one in Listings 1 and 2, request in Listing 6 and response in Listing 7 of the evaluation process according to the SBA-XACML policy evaluation semantics in Section 5.

Line 1 is the policy set *PS*. The policy set *ID* is *PS*1. It has two policies *P*1 and *P*2. *P*1 is ordered before *P*2. The policy combining algorithm is *Permit − Overrides*. *PS*1 has no reference to other policies. It

```
[1]. <Request xmlns="context:schema:os" xmlns:xsi="XMLSchema-instance">
[2].   <Subject  SubjectCategory="subject-category:access-subject">
[3].        <Attribute AttributeId="subject:subject-id" DataType="xml:string
    ">
[4].            <AttributeValue>Bob</AttributeValue>
[5].        </Attribute>
[6].   </Subject>
[7].   <Resource>
[8].        <Attribute AttributeId="resource:resource-id" DataType="xml:
    string">
[9].            <AttributeValue>BankService/deposit</AttributeValue>
[10].      </Attribute>
[11].  </Resource>
[12].  <Action>
[13].      <Attribute AttributeId="action:action-id" DataType="xml:string">
[14].           <AttributeValue>execute</AttributeValue>
[15].      </Attribute>
[16].  </Action>
[17].</Request>
```

Listing 3. XACML access request.

```
[1]. <Response>
[2]. <Result ResourceID="BankService/deposit">
[3]. <Decision>Permit</Decision>
[4]. <Status>
[5]. <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
[6]. </Status>
[7]. </Result>
[8]. </Response>
```

Listing 4. XACML access response.

```
[1].PS::=<PS1,{P1,P2},{P1>P2},{Permit-overrides},{},{},{{},{},{}}>
[2].P::=<P1,{R1,R2},{R1>R2},{Permit-overrides},{},{{Withdraw,Permit,{mailto,
    string,Customer_service@bank.com},{subject-id,string,subject-id},{
    resource-id,string,resource-id}}}          ,{{Jerry,Joe},{BankService/
    withdraw},{Any}}>
[3].R::=<R1,{{and,{string-equal,{ResourceAttributeDesignator,string,
    BankService/withdraw}},{string-equal,{SubjectAttributeDesignator,subject
    -id,string,Bob}}}},{{},{},{}},{Permit}>
[4].R::=<R2,{},{{},{},{}},{Deny}>
[5].P::=<P2,{R3,R4,R5},{R3>R4>R5},{Permit-overrides},{},{},{{},{},{}}>
[6].R::=<R3,{string-equal,{ResourceAttributeDesignator,string,BankService/
    deposit}},{{},{},{}},{Permit}>
[7].R::=<R4,{{and,{string-equal,{ResourceAttributeDesignator,string,
    BankService/deposit}},{string-equal,{SubjectAttributeDesignator,subject-
    id,string,Joe}}}},{{},{},{}},{Permit}>
[8].R::=<R5,{{and,{string-equal,{ResourceAttributeDesignator,string,
    BankService/deposit}},{string-equal,{SubjectAttributeDesignator,subject-
    id,string,Joe}}}},{{},{},{}},{Deny}>
```

Listing 5. SBA-XACML policy for a bank service.

```
[1].Rq1::=<{Bob},{BankService/Deposit},{execute}>
```

Listing 6. SBA-XACML access request.

```
[1].Rs::=<{permit},{}>
```

Listing 7. SBA-XACML response.

has no obligations to perform and the target subjects, resources and actions are any. Line 2 is the *Withdraw* policy. The policy *ID* is *P*1. It has two rules *R*1 and *R*2. *R*1 is ordered before *R*2. The rule combining algorithm is *Permit* − *Overrides*. *P*1 has one obligation to perform and the target subjects are *Bob* and *Jerry*, resource is *BankService/withdraw* and actions are any. Line 3 is the rule *R*1. The rule *ID* is *R*1. *R*1 has a set of conditions. The conditions are: the subject *ID* must be equal to *Bob* and the resource *ID* must be equal to *BankService/withdraw*. The target subjects, resources and actions are any. *R*1 has a *permit* effect. Line 4 is the rule *R*2. The rule *ID* is *R*2. *R*2 has no conditions. *R*2 has no target specified. *R*2 has a *deny* effect. Line 5 is the *deposit* policy. The policy *ID* is *P*2. It has three rules *R*3, *R*4 and *R*5. The precedence order is *R*3, *R*4 and *R*5. The rule combining algorithm is *permit* − *overrides*. *P*2 has no obligation to perform and the target elements are not defined. Line 6 is the rule *R*3. The rule *ID* is *R*3. *R*3 has one condition. The condition states that the resource *ID* must be equal to *BankService/Deposit*. The target subjects, resources and actions are not specified. *R*3 has a *permit* effect. Line 7 is the rule *R*4. The rule *ID* is *R*4. *R*4 has a set of conditions. The conditions are: the subject *ID* must be equal to *Joe* and the resource *ID* must be equal to *BankService/Deposit*. The target subjects, resources and actions are not specified. *R*4 has a *permit* effect. Line 8 is the rule *R*5. The rule *ID* is *R*5. *R*5 has a set of conditions. The conditions are: the subject *ID* must be equal to *Joe* and the resource *ID* must be equal to *BankService/Deposit*. The target subjects, resources and actions are not specified. *R*5 has a *deny* effect.

Listing 6 contains the generated SBA-XACML request. The request subject is equal to *Bob*, resource equal *BankService/Deposit* and action equal *execute*.

Based on the SBA-XACML policy evaluation semantics in Section 5 and its implemented algorithms in Section 6, the elaborated framework will evaluate the request *Rq1* in Listing 6 with respect to the based policy *PS1* presented in Listing 5. Since the evaluation of each semantics rule is based on evaluating its premises, we will describe the evaluation steps in order, by the premises of policy sets, policies and rules, as summarized in Table 5. To avoid repetition and for space limitation, we will present only the matching semantics rules that affect the final decision. The non matching ones will be ignored. The rules in Table 5 should be read from bottom to top as follows:

(1) The based policy is composed of a PolicySet *PS1*. It has *PCA* = {Permit-Overrides}, its target *TR* matches request *Rq1* as illustrated in (2) and it has a policy *P2* that evaluates to *Permit* as depicted in (3). Hence, based on the semantics Rule **9** that applies in this case, all the three premises are satisfied and the final decision is *Permit*.
(2) *PS1* has no target defined which means *TR* = {} or *TR* = {S = Any, R = Any, A = Any}. Rq1 = {Sr = Bob, Rr = BankService/Deposit, Ar = execute}. By applying semantics Rule **1**, Bob is a subset of Any, BankService/Deposit is a subset of Any and execute is a subset of Any, therefore *PS1* matches the request *Rq1*.
(3) *P2* is composed of three rules. It has *RCA* = Permit-Overrides, its target *TR* matches with the target of request *Rq1* as illustrated in (4) and it has a rule *R3* that evaluates to *Permit* as depicted in (5). Hence, based on the semantics Rule **6** that applies in this case, all the three premises are satisfied and the evaluation of *P2* with respect to *Rq1* is *Permit*.
(4) *P2* has no target *TR* defined which means *TR* = {S = Any, R = Any, A = Any}. Rq1 = {Sr = Bob, Rr = BankService/Deposit, Ar = execute}. By applying semantics Rule **1**, Bob is a subset of Any, BankService/Deposit is a subset of Any and execute is a subset of Any, therefore *P2* matches the request *Rq1*.
(5) *TR* of *R3* matches with the target of request *Rq1* as illustrated in (6). *R3* has one rule condition *RC* = {string-equal,{RAD, string, BankService/deposit}}, which means the resource requesting access must be equal to BankService/Deposit. *RC* = True because the Resource *R* of *Rq1* is equal to BankService/Deposit. The rule effect *RE* of *R3* is Permit (*RE* = Permit). Hence, based on the semantics Rule **3** that applies in this case, all the three premises are satisfied and the evaluation of *R3* with respect to *Rq1* is *Permit*.
(6) *R3* has no target defined which means TR = {S = Any, R = Any, A = Any}. Rq1 = {Sr = Bob, Rr = BankService/Deposit, Ar = execute}. By applying semantics Rule(**1**), Bob is a subset of Any, BankService/Deposit is a subset of Any and execute is a subset of Any, therefore *R3* matches the request *Rq1*.

The response to the request in Listing 6 against the based policy in Listing 5 is presented in Listing 7. The evaluation results of our approach always returns the same results given by XACML Sun PDP (Moses, 2011).

8. Experiments and performance analysis

In this section, we provide the experiments and results of the performance analysis comparing our scheme to the current approaches. We have implemented the SBA-XACML framework using PHP. Our experiments were carried out on a notebook running Windows XP SP3 with 3.50 GB of memory and dual core 2.8 GHz Intel processor. The experiments were performed at 100,000 tests each and the average number was calculated and used. They were conducted on both real world and synthetic policies to show the scalability and performance on very large ones. Synthetic policies are created in such a way that every policy and every rule in the policy set is evaluated to reach the final decision (i.e. taking always the worst case). The Synthetic policy sets range from 400 to 4000 rules which are split evenly over 100 policies. In

Table 5
Results of semantics-based policy evaluation.

$$\frac{(\{\{Bob\}\cap\{Any\})\neq\emptyset)\wedge(\{\{BankService/Deposit\}\cap\{Any\})\neq\emptyset)\wedge(\{\{execute\}\cap\{Any\})\neq\emptyset)}{\langle R3.TR,Rq1\rangle \underset{match}{\vdash} True \quad (\textbf{SemanticsRule}(\textbf{1}))} \quad (\textbf{6})$$

$$\frac{(\langle R3.TR,Rq1\rangle \underset{match}{\vdash} True)\wedge(R3.RC=True)\wedge(R3.RE=Permit)}{\langle R3,Rq1\rangle \underset{eval}{\longrightarrow} Permit \quad (\textbf{SemanticsRule}(\textbf{3}))} \quad (\textbf{5})$$

$$\frac{(\{\{Bob\}\cap\{Any\})\neq\emptyset)\wedge(\{\{BankService/Deposit\}\cap\{Any\})\neq\emptyset)\wedge(\{\{execute\}\cap\{Any\})\neq\emptyset)}{\langle P2.TR,Rq1\rangle \underset{match}{\vdash} True \quad (\textbf{SemanticsRule}(\textbf{1}))} \quad (\textbf{4})$$

$$\frac{(P2.RCA=Permit-Overrides)\quad\wedge(\langle P2.TR,Rq1\rangle \underset{match}{\vdash} True)\wedge(\langle R3,Rq\rangle \underset{eval}{\longrightarrow} Permit)}{\langle P2,Rq\rangle \underset{eval}{\longrightarrow} Permit \quad (\textbf{SemanticsRule}(\textbf{6}))} \quad (\textbf{3})$$

$$\frac{(\{\{Bob\}\cap\{Any\})\neq\emptyset)\wedge(\{\{BankService/Deposit\}\cap\{Any\})\neq\emptyset)\wedge(\{\{execute\}\cap\{Any\})\neq\emptyset)}{\langle PS1.TR,Rq1\rangle \underset{match}{\vdash} True \quad (\textbf{SemanticsRule}(\textbf{1}))} \quad (\textbf{2})$$

$$\frac{(PS1.PCA=Permit-Overrides)\wedge(\langle PS1.TR,Rq1\rangle \underset{match}{\vdash} True)\wedge(\langle P2,Rq1\rangle \underset{eval}{\longrightarrow} Permit)}{\langle PS1,Rq1\rangle \underset{eval}{\longrightarrow} Permit \quad (\textbf{SemanticsRule}(\textbf{9}))} \quad (\textbf{1})$$

order to be able to exhaust the entire policy set, we specified (1) a policy combining algorithm *Deny-Overrides*, (2) rule combining algorithm *Deny-Overrides* for each policy, (3) the deny rule as the last rule in the policy and (4) non empty target element. Please note that moderate specification (i.e. decision is taking early without checking all the rules) of the synthetic policies will lead to better performance. We compare our proposed framework to the commercial XACML engine Sun PDP (Moses, 2011) and XEngine (Liu et al., 2008).

The processing time of Sun PDP consists of XACML policy loading, request loading and request evaluation to provide the decision. There is no pre-processing time for Sun PDP. As for XEngine, the pre-processing time consists of policy loading, numericalization and normalization, while processing time consists of request loading, numericalization and evaluation to provide the decision. Regarding our approach, the pre-processing consists of converting policy set from XACML to SBA-XACML, which is optional and executed only once when deploying the policies. The processing time includes (1) accepting a request and converting it to SBA-XACML, (2) loading policies and (3) evaluating the request to providing the decision. We repeated this policy evaluation process for 100,000 different requests with and without the pre-processing procedures and provided the average evaluation time of synthetic and real world policies.

We chose not to use the experiments methodology used by XEngine because it does not reflect real world environments. Their tools and experiments show that all the requests (i.e. up to 100,000 requests) are received, converted and loaded in the memory at the same time, then evaluated against the already loaded policies. Again, as aforementioned, such assumption does not always hold since requests can be received from different parties at variant time–space.

8.1. Policy evaluation experimental results

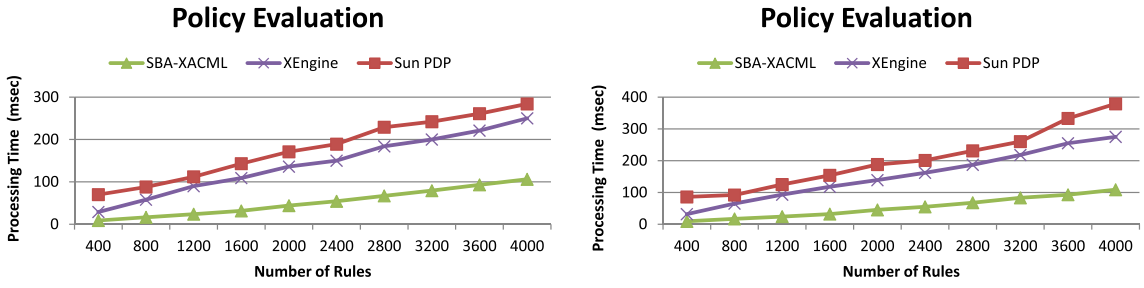
In the following, we discuss the experimental results for single-valued and multi-valued requests on both synthetic and real world policies. Fig. 2 shows the results for synthetic policy evaluations for single-valued and multi-valued requests. For single-valued requests, Fig. 2a shows that our approach is faster than both the XEngine and Sun PDP by 3.2 and 8 times respectively for policy sets with 400 rules, and by 2.4 and 2.7 times faster for policy sets with 4000 rules. For multi-valued requests, Fig. 2b shows that our approach is faster than both the XEngine and Sun PDP by 3.5 and 9.4 times respectively for policy sets with 400 rules, and by 2.5 and 3.5 times faster for policy sets with 4000 rules.

Fig. 3 contains three real world policies. We included the number of rules in each policy set, the average pre-processing time (conversion time) for SBA-XACML and XEngine, the average processing time for single-valued and multi-valued requests for SBA-XACML, XEngine and Sun PDP. Moreover, Fig. 4 shows graphically the results for real world policy evaluation for single-valued and multi-valued requests. Fig. 4a explores that our approach is 8 times faster than XEngine and 39 times faster than the Sun PDP for small policies with less than 10 rules, while it is 7.6 faster than XEngine and 3 times faster than Sun PDP on policies with 300 rules. Fig. 4b shows that our approach is faster by 2.4 times and 6.3 than XEngine and Sun PDP respectively for a policy set with 298 rules, while it is 3.5 times and 15 times faster than XEngine and Sun PDP respectively for small policies with less than 10 rules.

8.2. Policy evaluation experimental results including pre-processing

In the following, we discuss the conversion time from XACML to SBA-XACML with respect to the XEngine conversion time. The conversion time is referred to as pre-processing time in (Liu et al., 2008). The conversion procedure is optional in our approach and required only during the policy deployment. However, the XEngine approach requires this step because the conversion includes numericalization and normalization and it is the core of their proposal to improve the evaluation response time. Fig. 5a and b show the conversion time for synthetic and real world policies respectively. Assuming the Synthetic policies are in XACML, the XEngine pre-processing time consumes 2.3 times more than our approach for a policy set with 4000 rules, and 8.5 times more for a policy set with 400 rules. For real policies, the XEngine requires 8 times more than our approach for policy sets with 298 rules, and 10 times more for policy sets with less than 10 rules.

Fig. 6 shows the results of the overall processing and pre-processing (i.e. conversion) time of single-valued and multi-valued

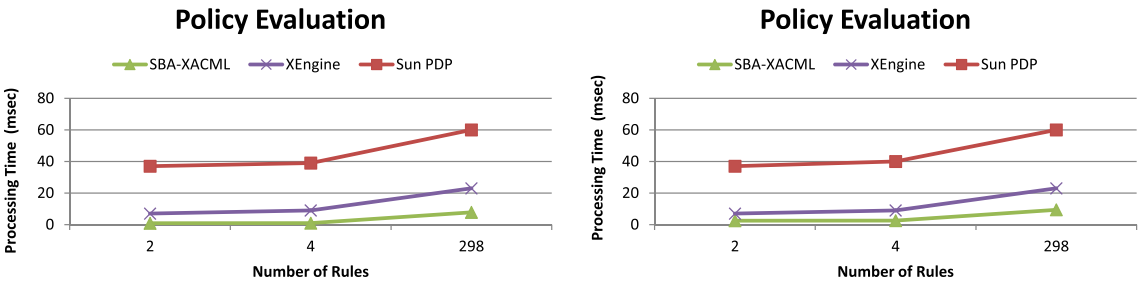


(a) Single-valued Request (b) Multi-valued Request

Fig. 2. synthetic policy evaluation.

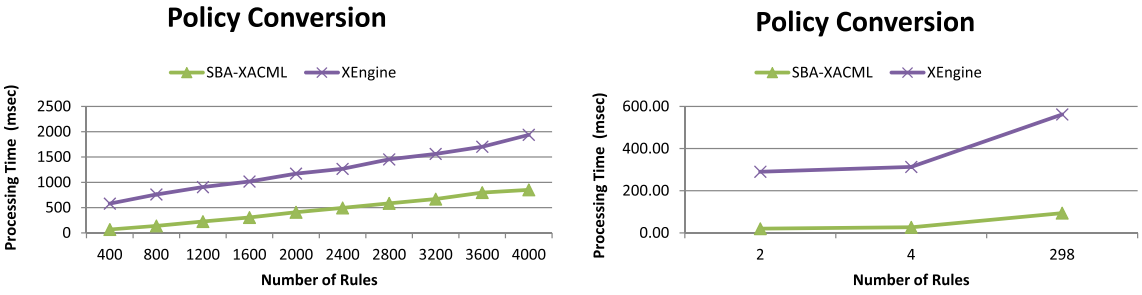
Policy	#Rules	Conversion Time (msec)		Average Processing Time (msec)					
				Single-valued Requests			Multi-valued Requests		
		SBA-XACML	XEngine	SBA-XACML	XEngine	Sun PDP	SBA-XACML	XEngine	Sun PDP
IIIA027	2	20	290	1	7	37	2	7	37
IIIA028	4	27	313	1	9	39	3	9	40
Continue-a	298	94	562	8	23	60	9	23	60

Fig. 3. Experimental results on real-world XACML policies.



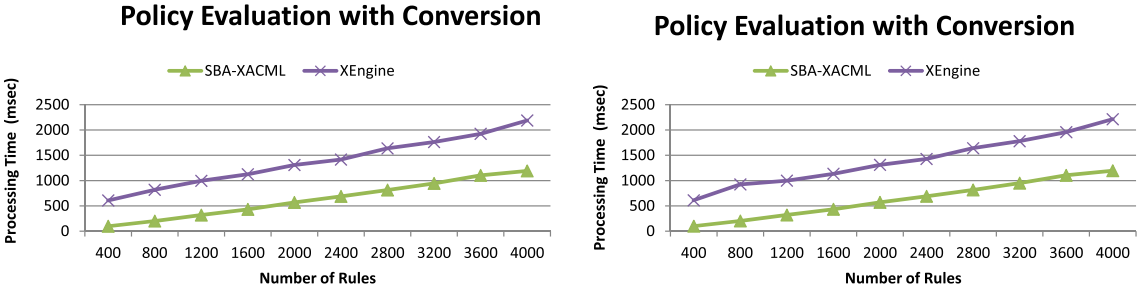
(a) Single-valued Request (b) Multi-valued Request

Fig. 4. Real policy evaluation.



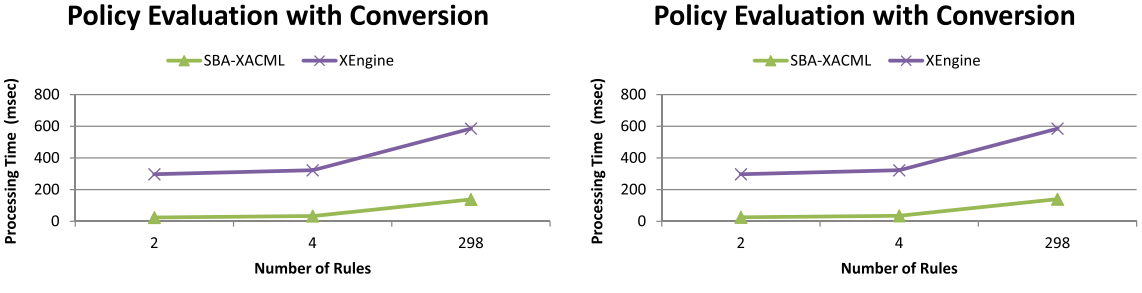
(a) Synthetic Policies (b) Real Policies

Fig. 5. Policy conversion.



(a) Single-valued Request (b) Multi-valued Request

Fig. 6. Synthetic policy evaluation with conversion.



(a) Single-valued Request (b) Multi-valued Request

Fig. 7. Real policy evaluation with conversion.

requests against synthetic policies ranging from 400 to 4000 rules. Fig. 6a shows that our approach outperformed the XEngine by 6 times for small policy sets with 400 rules, and by 2 times for large policy sets with 4000 rules. Fig. 6b illustrates that our approach outperformed the XEngine by 6.1 times for small policy sets with 400 rules, and by 2 times for large policy sets with 4000 rules.

Fig. 7 shows the results for the overall processing and pre-processing (i.e. conversion) for single-valued and multi-valued requests against real policies ranging from 2 to 298 rules. Fig. 7a shows that SBA-XACML outperformed the XEngine by 9 times for small policy sets with less than 10 rules and by 4.2 times for medium size policy sets with 300 rules. Fig. 7b illustrates that our approach outperformed

the XEngine by 9 times for small policy sets with less than 10 rules, and by 4 times for policy sets with 300 rules.

9. Conclusion and future work

This paper addressed problems related to the performance of real-time evaluation of composite and complex XACML policies for accessing Web services. In this context, the contribution of this work is the elaboration of a novel set-based scheme called SBA-XACML that provides efficient policy evaluation and decision process. Our proposition constitutes of (1) a mathematical intermediate representation of policies based on set theory that maintains the same XACML structure and accounts for all its elements and their sub elements including rule conditions, obligations, policy request and policy response; and (2) a formal semantics that takes advantage of the mathematical operations to provide efficient evaluation of policy elements and constructs through deductive logic and inference rules. Our proposed approach improves the related literature by reducing the complexity of the policies and overhead of real-time policy evaluation. Moreover, it offers the first complete formal representation of XACML constructs, which opens the door for potential analysis on the policy meaning. Furthermore, it maintains the same architecture of the industrial standard XACML Sun PDP (Moses, 2011) and respects the major properties and assumptions of real-life environments in terms of remote policy loading upon need and disjoint reception of requests from distributed services.

The aforementioned theoretical outcome are also supported by developing practical algorithms, modules and experiments using real-life and synthetic policies. The proposed algebra language including its compiler have been implemented in a module that provides automatic and optional conversion from XACML to SBA-XACML constructs. Moreover, the proposed semantics have been realized into algorithms for policy evaluation and decision making. Finally, the conducted experiments explore that SBA-XACML evaluation of large and small sizes policies provide better performance than Sun PDP (Moses, 2011) and its corresponding ameliorations in the literature (Liu et al., 2008; Marouf et al., 2011; Ngo et al., 2013; Pina Ros et al., 2012), by a factor ranging between 2.4 and 15 times faster depending on policy size. To download and get additional information about the developed framework and experiments, please visit the following link: http://www.azzammourad.org/#projects.

There are many directions for future work to be built on top of our approach. First, we can benefit from SBA-XACML to potentially elaborate policy analysis semantics based on the meaning of rules for detecting access flaws, conflicts and redundancies. This problem is raised when having complex policies with hundred and even thousands of rules, or when several XACML policies from different Web services (Ayoubi et al., 2013; Karakoc & Senkul, 2009; Tout et al., 2013; Yahyaoui et al., 2012) are grouped and composed together, where contradiction between combining algorithms may occur. In this regard, few approaches have been proposed addressing XACML policy composition and analysis (Rao et al., 2009; Bonatti, Vimercati, & Samarati, 2002; Fisler, Krishnamurthi, Meyerovich, & Tschantz, 2005; Kolovski et al., 2007; Mazzoleni et al., 2006; Tschantz & Krishnamurthi, 2006; Wijesekera & Jajodia, 2003). However, none of them addressed the aforementioned problems. Furthermore, few works (Karakoc & Senkul, 2009; Khosrowshahi Asl et al., 2014) have been addressing issues related to composing or grouping Web services while taking into consideration several real-time and context-aware factors. Detecting semantics-based access contradictions and conflicts between the candidates Web services, which would influence the grouping decision, will be an interesting expansion to the current approach. It will be also interesting to study the impact of such detection on the overall grouping decision. Moreover, several ameliorations can be performed on the proposed algorithms in order to provide more efficient evaluation process.

Acknowledgment

This work is supported by the Lebanese American University(LAU) and CNRS, Lebanon.

References

Ayoubi, S., Mourad, A., Otrok, H., & Shahin, A. (2013). New XACML-AspectBPEL approach for composite web services security. *International Journal of Web and Grid Services, 9*(2), 127–145.

Bhalla, N. & Kazerooni, S. (2007). Web services vulnerabilities. <http://www.blackhat.com/presentations/bh-europe-07/Bhalla-Kazerooni/Whitepaper/bh-eu-07-bhalla-WP.pdf>.

Bonatti, P., Vimercati, S. D. C. D., & Samarati, P. (2002). An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISS), 5*(1), 1–35.

Fisler, K., Krishnamurthi, S., Meyerovich, L. & Tschantz, M. (2005). Verification and change impact analysis of access-control policies. In *Proceedings of 27th international conference on software engineering (ICSE)* (pp. 196–205).

Karakoc, E., & Senkul, P. (2009). Composing semantic web services under constraints. *Expert Systems with Applications, 36*(8), 11021–11029.

Khosrowshahi Asl, E., Bentahar, J., Mizouni, R., Khosravifar, B., & Otrok, H. (2014). To compete or cooperate? This is the question in communities of autonomous services. *Expert Systems with Applications, 41*(2), 4878–4890.

Kolovski, V., Hendler & J. Parsia B. (2007). Analyzing web access control policies. In *Proceedings of the 16th international conference on world wide web (WWW'07)* (pp. 677–686).

Liu, A.X., Chen, F., Hwang, J. & Xie, T. (2008). XEngine: a fast and scalable XACML policy evaluation engine. In *Proceedings of the SIGMETRICS international conference on measurement and modeling of computer systems* (pp. 265–276).

Mazzoleni, P., Bertino, E. & Crispo, B. (2006). XACML policy integration algorithms: not to be confused with XACML policy combination algorithms!. In *Proceedings of the 11th ACM symposium on access control models and technologies (SACMAT2006)* (pp. 219–227).

Marouf, S., Shehab, M., Squicciarini, A., & Sundareswaran, S. (2011). Adaptive reordering and clustering based framework for efficient XACML policy evaluation. *IEEE Transactions on Services Computing, 4*(4), 300–313.

Moses, T. (2011). OASIS eXtensible Access Control Markup Language(XACML), OASIS Standard 2.0. <http://www.oasis-open.org/committees/xacml/>.

Mourad, A., Ayoubi, S., Yahyaoui, H., & Otrok, H. (2012). A novel aspect-oriented BPEL framework for the dynamic enforcement of web services security. *International Journal of Web and Grid Services, 8*(4), 361–385.

Ngo, C., Makkes, M., Demchenko, Y. & de Laat, C. (2013). Multi-data-types interval decision diagrams for XACML evaluation engine. In *Proceedings of the 11th international conference on privacy, security and trust (PST 2013)* (pp. 257–266).

Pagan, F. G. (1981). *Formal specification of programming languages*. Prentice-Hall, Inc..

Plotkin, G. D. (2004). A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 17–139.

Pina Ros, S., Lischka, M. & Gómez Mármol, F. (2012). Graph-based XACML evaluation. In *Proceedings of the 17th ACM symposium on access control models and technologies (SACMAT12)* (pp. 83–92).

Rao, P., Lin, D., Bertino. E., Li, N. & Lobo, J. (2009). An algebra for fine-grained integration of XACML policies. In *Proceedings of the 14th ACM symposium on access control models and technologies (SACMAT2009)* (pp. 63–69).

Slonneger, K., & Kurtz, B. L. (1995). *Formal syntax and semantics of programming language: A laboratory based approach*. Addison-Wesley Publishing Company, Inc..

Tout, H., Mourad, A., & Otrok, H. (2013). XrML-RBLicensing approach adopted to the BPEL process of composite web services. *Journal of Service Oriented Computing, 7*(3), 217–230.

Tschantz, M. & Krishnamurthi, S. (2006). Towards reasonability properties for access-control policy languages. In *Proceedings of the eleventh ACM symposium on access control models and technologies (SACMAT2006)* (pp. 160–169).

Wang, M., Wang, H., Xu, D., Kit Wan, K., & Vogel, D. (2004). A web-service agent-based decision support system for securities exception management. *Expert Systems with Applications, 27*(3), 439–450.

Wijesekera, D., & Jajodia, S. (2003). A propositional policy algebra for access control. *ACM Transactions on Information and System Security (TISS), 6*(2), 286–325.

Yahyaoui, H., Mourad, A., AlMulla, M., Yao, L., & Sheng, Q. Z. (2012). A synergy between context-aware and AOP to achieve highly adaptable web services. *Journal of Service Oriented Computing, 6*(4), 379–392.