Computers and Electrical Engineering 52 (2016) 65–79

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

From model-driven specification to design-level set-based analysis of XACML policies☆

Azzam Mourad[a,*], Hanine Tout[b], Chamseddine Talhi[b], Hadi Otrok[c], Hamdi Yahyaoui[d]

[a] Department of Computer Science & Mathematics, Lebanese American University, Lebanon
[b] Department of Software Engineering & Information Technology, École de Technologie Supérieure, Canada
[c] Department of Computer Engineering, Khalifa University of Science, Technology & Research, United Arab Emirates
[d] Department of Computer Science, Kuwait University, Kuwait

A R T I C L E I N F O

Article history:
Received 27 May 2015
Revised 23 September 2015
Accepted 25 September 2015
Available online 17 October 2015

Keywords:
Web services security
XACML policies
Model-driven specification
Design-level analysis
Logical deductions
Access control

A B S T R A C T

With lot of hype surrounding policy-based computing, XACML (eXtensible Access Control Markup Language) has become the widely used de facto standard for managing access to open and distributed service-based environments like Web services. However, like any other policy language, XACML has complex syntax, which makes the policies specification process both time consuming and error prone, especially with large size policies that govern complex systems. Moreover, with the diversity of rules and conditions, hidden conflicts, redundancies and access flaws are more likely to arise, which expose Web services to security breaches at runtime. This paper proposes a UML profile that allows systematic model-driven specification of XACML policies to resolve the complexity of policies designation. Based on mathematical sets that explore the rules meanings, the paper provides also a design-level analysis to detect anomalies in the specified policies, prior to their enforcement in the system. A real life case study demonstrates the feasibility and efficiency of the proposition.

1. Introduction

Web services technology is undoubtedly revolutionizing distributed computing with its phenomenal support to expose, integrate and enable access to open, distributed and heterogeneous services ubiquitous over the internet. Yet, the vistas opened by this technology cannot hide its underlying security holes. With infinite accessibility to services over the internet and the evolution of cyber-crimes, access control is still one of the biggest concerns in the development of Web services. Hereof, policy-based computing [1–3] has gained considerable attention in controlling access to different systems including Web services. Particularly, XACML [4] has become the most widely used de facto standard for specifying and enforcing access control policies in this field [5–7].

However, like any other policy language, XACML has too low-level and complex syntax, which makes the policies specification process time consuming and error-prone. These problems emerge mainly when defining and combining many policies to govern access to complex systems like Web services composition [8]. Accordingly, some researchers [9–12] have proposed UML profiles

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Associate Editor Dr. S. Rajavelu.
* Corresponding author. Tel.: +96170755517.
E-mail address: azzam.mourad@lau.edu.lb, azzammourad@gmail.com (A. Mourad).

http://dx.doi.org/10.1016/j.compeleceng.2015.09.021
0045-7906/© 2015 Elsevier Ltd. All rights reserved.

to offer high-level graphical modeling mechanism for policies specification. UML profile [13] is a lightweight UML extension mechanism that allows creating a domain specific language. Existing approaches have proved the capabilities of such mechanism to define different access control models. Yet, these works lack a full coverage of XACML elements, which makes the proposed profiles unable to support all policies that can be expressed in this language.

Moreover, with the diversity of rules and conditions, anomalies are more likely to arise. Rules can overlap when single access request matches multiple rules. They may also conflict with each other in case these rules overlap and yield different decisions. Such conflicts in XACML policies are critical since they would allow unauthorized access to critical services or denial to legitimate entities. Redundancy is another type of anomalies in XACML policies. When an access request matches different rules with the same effect, these rules are considered as redundant. Redundancy affects mainly the performance of the access control decision-making process [14], which is influenced by the number of rules to be evaluated. The third critical anomaly in XACML policies is access flaw, which arises with badly defined and ordered rules and/or policies that allow users to gain accidental access to particular resources. Besides the achievements in the formal verification [15] of complex systems, several approaches [16–18] have proposed analysis mechanisms for XACML policies, few of them were able to detect all three aforementioned anomalies, yet not at the design level. In addition, none of these approaches is based on logical deductions that explore the meaning of the rules declared in the policies.

In this paper, we present a new approach that aims to tackle these issues. To resolve the complexity of policies specification, we propose a UML profile that covers all the elements of XACML and allow systematic model-driven specification of standard XACML policies. In addition, we devise set-based algorithms for design-level and logical analysis of the specified policies in order to detect conflicts, redundancies and access flaws within and among them, prior to their enforcement in the system. The detection algorithms are based on structured sets constructs automatically extracted from the policies models. These sets are analyzed based on mathematical relations that reflect the semantics of the rules declared in the model. A list of existing anomalies is generated to locate them and eventually help resolving them. The evaluation of the policies [19–21] is done by a decision-making engine that we presented in previous work [22] and proved its ability to accelerate the evaluation process over other existing engines [4,23]. Controlling access to single independent Web service implies direct call to this engine, while in case of several services composed in a business process, our approach involves an aspect-oriented policy enforcer. The latter is responsible of automatically generating calls for the evaluation engine and dynamically weaving them in the Web services composition to manage the critical resources declared in the model.

This work offers the following contributions:

- Providing UML profile that fully supports all XACML language constructs to allow model-driven specification of XACML policies. The profile can be systematically applied to design XACML access control policies for Web services.
- Devising algorithms for design-level and logical analysis of XACML policies in order to detect anomalies like conflicts, redundancies, and access flaws. Moreover, identifying such anomalies between policies belonging to different Web services allows to appropriately select/eliminate the ones participating in the composed business model. The analysis is done on set-based constructs, which are automatically generated from the policies UML models.
- Delivering policy enforcer that can automatically extracts critical resources from the designated policies model, generate calls for the evaluation engine and dynamically enforce these calls in the system to control access to these resources.

The rest of the paper is organized as follows. In Section 2, we study existing relevant approaches, while in Section 3 we give an overview of our proposition. In Section 4, we present the elaborated UML profile. In Section 5, we explain the intermediate set-based constructs for the XACML UML models. We present the logical detection mechanism and relevant algorithms in Section 6. In Section 7, we explain the policies enforcement process. In Section 8, we devote a case study to illustrate the proposed approach. Finally, we conclude the paper and draw future research directions in Section 9.

2. Related work

We present in this section existing works for model-driven specification techniques and analysis mechanisms for XACML policies. We also discuss their limitations to emphasize on the contributions offered by our proposition.

2.1. Model-driven specification of XACML policies

A very few works have been proposed for mode-driven specification of XACML policies. Jin [9] has proposed model-driven architecture to build role based access control (RBAC) model. To address the complexity of XACML XML-based documents, they proposed a UML profile that provides UML notations to ease the specification of XACML RBAC applications. In the proposed approach designers are able to create RBAC XACML specification platform-independent models based on the access control requirements along with the application models. However, the proposed profile does not cover important elements of XAMCL like obligations and advices.

Busch et al. [10] argued that XML syntax of XACML makes the process of policies specification difficult and error-prone and thus they proposed a UML-based notation to offers graphical modelling of security properties for Web applications. The authors leverage the UML-based Web Engineering using the extension mechanisms UML profile extension mechanism. Yet, again the proposed approach does not support obligations within the policy sets, policies and rules.

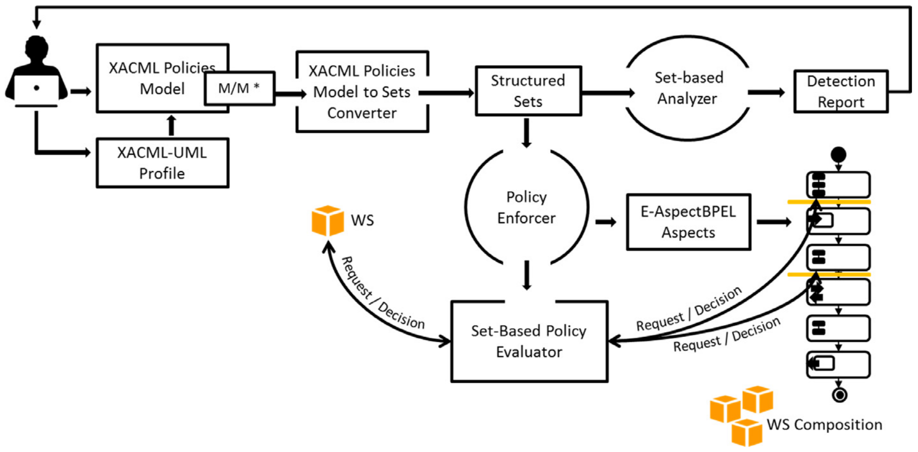

Fig. 1. Proposed architecture.

A few initiatives have been proposed in this area exploiting the UML profile extension mechanism provided by UML. Existing profiles notations presented in these approaches did not cover all the constructs of XACML, hence their profiles cannot support all policies that can be designated with this language. Besides that these approaches do not rely on the latest version of XACML. Per contra, we present in this paper a UML profile that covers all the elements of XACML including policy sets, policies, rules, targets, conditions, obligations and advices, hence offering the ability to design any policy that can be expressed via the standard XACML and conforms with the latest version of this language.

2.2. Analysis mechanisms for XACML policies

For the analysis of XACML policies, several approaches have been proposed. Huonder [18] proposed an approach that focuses on the detection and resolution of conflicts in XACML policies. To detect conflicts, the proposed approach maps each target to the n-dimensional space and locate all overlapping policies with different effects. With such technique, every policy is considered as a set of attributes and a non-empty intersection in all dimensions between policies defines a conflict among them. Yet, the proposed approach did not address access flaws and redundancies anomalies, does not support the combining algorithms between XACML policies and excludes the rule conditions from the analysis.

Kolovski et al. [16] proposed an approach based on description logics (DL), which are a decidable fragment of First-Order logic. To verify policies, the authors use the existing DL verifiers. Their analysis process can discover redundancies at the rule level. However, they do not address access flaws and do not support many important criteria like multi-subject requests, complex attribute functions, rule Conditions and Only-One-Applicable combining algorithm.

Rao et al. [17] introduced algebra for fine-grained integration that supports specification of a large variety of integration constraints. They introduced a notion of completeness and proved that their algebra is complete with respect to this notion. Their approach, however, does not support obligations and rule conditions and focuses on combined policies, unlike ours which offers the ability to analyze policy sets individually and after integration from different parties.

Several researchers have proposed analysis mechanisms to detect anomalies in XACML policies. With respect to these approaches, our work mainly differs in different aspects. First, it offers the ability to detect conflicts, flaws and redundancies, where to the best of our knowledge none of the existing approaches address these three anomalies. Second, the analysis is done at design level before enforcing policies in the system, assisting designers to locate anomalies and hence avoid their propagation. Third, our analysis mechanism is based on logical deductions that explore the meaning of the rules declared in the policies. Finally, it does not make any assumption or exclude any element of the standard XACML language.

3. Approach overview

The overall architecture of our approach is illustrated in Fig. 1 which includes a UML Profile, an Analysis Module, a Policy Enforcer and an Evaluation Module. Rather than struggling with the low level and complex syntax of XACML, the user creates a UML model (M in Fig. 1) where the proposed profile (see Section 4), can be systematically applied to design access control policies. The user can also analyze the models to detect conflicts, redundancies and access flaws and get the corresponding analysis report. The set-based constructs, which we interpret in Section 5, get automatically generated to allow the logical analysis based on the declared rules. The analysis process is conducted using the analysis module that embeds the algorithms in Section 6. The generated report details and locates the problematic rules in the policies model in order to assist the designer to select the appropriate strategy that best resolves the detected anomalies. XACML can resolve conflicts only through its combining algorithms, yet does not address flaws and redundancies and does not support design level solutions. Therefore resolving these anomalies require the definition of new customized strategies, which include cutting out rules overruled by others or selecting policies

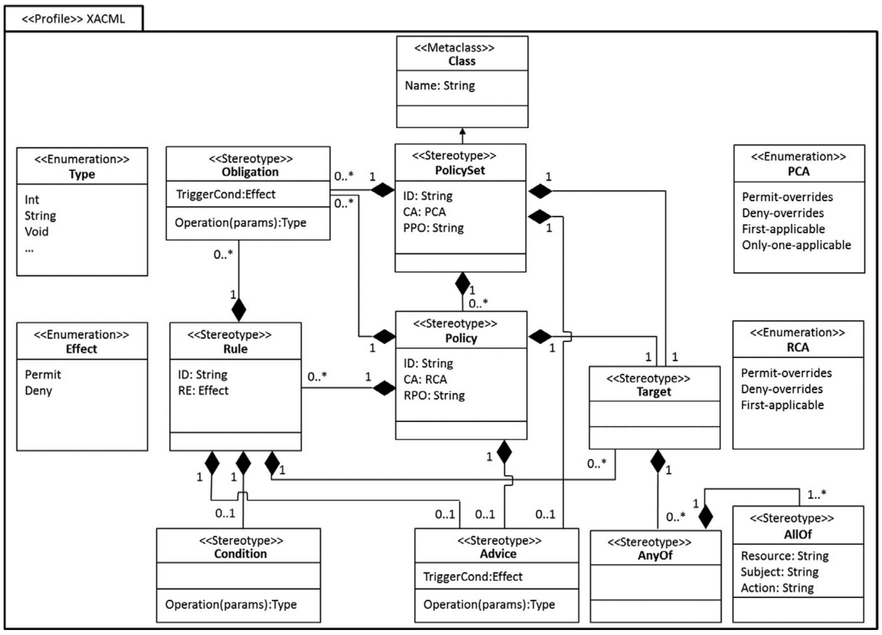

Fig. 2. XACML-UML profile.

from different Web service providers that do not cause anomalies when combined together. The automation of such strategies forms part of our future work. The policies model is modified accordingly (M* in Fig. 1) and reanalyzed.

In addition, the approach offers the ability to evaluate the policies. The evaluation is done by a decision-making engine that we presented in previous work [22], where it proved its qualification to accelerate the decision-making process over other engines in the literature. Controlling access to single independent Web service implies direct call to this engine. Whereas for a business process of composite Web services, our approach adds a policy enforcer that can automatically generate calls for the evaluation engine and dynamically weave them in the composition to control access to the critical resources declared in the policy set model. The calls are created based on AspectBPEL [1], an Aspect-Oriented Programming language that offers the ability to define security, business and context-aware aspects as independent components and automatically integrate them in the Web services composition [24–28].

4. UML profile for model-driven specification of XACML policies

In this section, we present the UML profile that we are proposing to provide model-driven technique to design XACML policies. In the sequel, we interpret the profile elements illustrated in Fig. 2. To remove any ambiguity, we used as much as possible the same names of the elements in the XACML language constructs.

We define the appropriate stereotypes, tagged definitions, operations and enumerations to cover all the elements of the latest version of XACML 3.0 that includes new elements and definition capabilities over its predecessors. A `PolicySet`, which extends the Metaclass `Class`, is a container of one or many `Policies`. It has an identifier `ID`, a policy proceeding order `PPO` that determines the order between its policies, and one of the policies combining algorithms `PCA` (i.e., Permit-overrides, Deny-overrides, First-applicable, and Only-one-applicable). These algorithms are used in XACML to resolve decision application problems between policies.

A `Policy` has also an identifier `ID` and may include many `Rules`. It has `RPO` to define precedence order among them and one of the rule combining algorithms `RCA` (i.e., Permit-overrides, Deny-overrides, and First-applicable) to resolve decision problems between its rules. Each rule can has a `Condition`, which is a function that should be validated before applying the rule.

`PolicySet`, `Policy`, and `Rule` can be all associated with `Targets`, `Obligations` and `Advices`. A `Target` identifies the action that a subject can exercise on certain resource, where in our case the action is an invoke and the resource is a service offered by partner Web service. The `Obligation` is a an action `Operation(params)` to be taken when certain trigger condition `TriggerCond` is met. A rule effect `ER` is the decision whether to permit or deny access. Finally, the `Advice` an analogous to `Obligation`, yet its common use is to detail why particular subject was denied access to certain resource.

5. Set-based constructs for XACML policies model

The models are automatically converted into structured set-based constructs using the convertor module. The conversion to such intermediate representation allows at later stage the logical analysis of policies based on the semantics of their rules. The

```
PS::={PS1,{P1},{},{permit-overrides},{},{},{{},{},{}}}
P::={P1,{R1,R2},{R1>R2},{premit-overrides},{},{},{{},{},{}}}
R::={R1, {{string-equal,{ResourceAttributeDesignator,string,getFinancialData}},
      {string-equal,{SubjectAttributeDesignator,subject-id,string,Admin}}},
      {{},{},{}},{Permit}}
R::={R2, {{and,{string-equal,{ResourceAttributeDesignator,string,getFinancialData}},
      {string-equal,{SubjectAttributeDesignator,subject-id,string,Employee}}}},
      {{},{},{}},{Deny}}
```

Fig. 3. Set-based policy.

convertor takes the policies UML model defined by the user after applying the proposed profile, then parses the elements in the model and generates the appropriate sets, which are defined as follows:

$$PS = \{ID, SOP, PPO, PCA, OBLs, ADs, TAR\} \text{ (Construct1)}$$

The first generated set is the policy set *PS*, it includes its identifier *ID*, references to the set of policies it contains *SOP*, the order between the policies *PPO*, the combing algorithm *PCA*, sets of obligations *OBLs* and Advices *ADs* if any, and finally the target *TAR* defined as another set of subject *S*, resource *RES* and action *A*.

$$TAR = \{S, RES, A\} \text{ (Construct2)}$$

Next, a set for policies is generated. Other than the policy *ID*, this set includes references to the set of corresponding rules *SOR*, precedence order between them *RPO*, combining algorithm *RCA*, sets of obligations *OBLs* and Advices *ADs* if any, and the target *TAR*.

$$P = \{ID, SOR, RPO, RCA, OBLs, ADs, TAR\} \text{ (Construct3)}$$

Finally comes the rule set *R*, which includes *ID*, condition *C*, sets of obligations *OBLs* and Advices *ADs* if any, target *TAR*, and rule effect *E*.

$$R = \{ID, C, OBLs, ADs, TAR, E\} \text{ (Construct4)}$$

C is a function to be evaluated against the target elements (i.e., subject, resource and action), which is defined as

$$C = \{Operation, \{params\}\} \text{ (Construct5)}$$

OBLs is a set of obligations *OBL*, where each is defined as

$$OBL = \{Operation, \{params\}\} \text{ (Construct6)}$$

and *ADs* is a set of advices *AD*, also defined in the same way.

$$AD = \{Operation, \{params\}\} \text{ (Construct7)}$$

Example. Consider a policy set *PS*1 that consists of a single policy *P*1 with a combining algorithm $PCA = permit - overrides$. *PS*1 does not refer to any other policy and does not have a target or any obligation.

Policy *P*1 defines two rules *R*1 and *R*2, with a precedence order $PRO = R1 > R2$. It applies a combining algorithm of $RCA = permit - overrides$ between its rules. *P*1 has no target and no obligations.

The first rule in this policy is *R*1, which has an effect $E = permits$ conditioned by allowing access to *getFinancialData* only for *Admin*. Whereas the second rule *R*2 has an effect $E = deny$ to prohibit access to *getFinancialData* only by an *Employee*. The set-based syntax of the policy set *PS*1, the policy *P*1 and both rules *R*1 and *R*2 is illustrated in Fig. 3.

The first rule in this policy is *R*1, which has an effect $E = permits$ conditioned by allowing access to *getFinancialData* only for *Admin*. Whereas the second rule *R*2 has an effect $E = deny$ to prohibit access to *getFinancialData* only by an *Employee*. The set-based syntax of the policy set *PS*1, the policy *P*1 and both rules *R*1 and *R*2 is illustrated in Fig. 3.

6. Set-based analysis

Detecting conflicts, access flaws and redundancies is done based on set theory that explores the meaning of the rules in each policy to the generated sets. The relations between rules are mapped to sets mathematical relations (i.e. inclusion, intersection, union, etc.) in order to express their meanings. We present in what follows the algorithms that realize the analysis process, which is done at three levels. In the first level, the rules are analyzed using Algorithm 1. In the second level, we devote Algorithm 2, which calls two other Algorithm 3 and 4 to detect anomalies within each policy and among combined policies respectively. Finally, in the third layer, Algorithm 5 analyzes the policy set.

Algorithm 1 Rule analyzer (R_1, R_2).

```
1:  // Extract elements for analysis
2:  //Get targets from both rules,
3:  $TAR_1 := R_1.getTarget();$
4:  $TAR_2 := R_2.getTarget();$
5:  // Extract subjects
6:  $S_1 := TAR_1.getSubjects();$
7:  $S_2 := TAR_2.getSubjects();$
8:  // Extract resources
9:  $RES_1 := TAR_1.getResources();$
10: $RES_2 := TAR_2.getResources();$
11: // Extract actions
12: $A_1 := TAR_1.getActions();$
13: $A_2 := TAR_2.getActions();$
14: // Get rules conditions
15: $C_1 := R_1.getConditions();$
16: $C_2 := R_2.getConditions();$
17: // Get rule effect
18: $E_1 := R_1.getEffect();$
19: $E_2 := R_2.getEffect();$
20: // Start analysis
21: $SFF := S_2$.areSubsetsorEqualto($S_1$);
22: $RESF := RES_2$.areSubsetsorEqualto($RES_1$);
23: $AFF := A_2$.areSubsetsorEqualto($A_1$);
24: $CFF := C_2$.areSubsetsorEqualto($C_1$);
25: $SSF := S_1$.intersctWith($S_2$);
26: $RESSF := RES_1$.intersctWith($RES_2$);
27: $ASF := A_1$.intersctWith($A_2$);
28: $CSF := C_1$.intersctWith($C_2$);
29: if ($SFF == True$ && $RESFF == True$ && $AFF == True$ && $CFF == True$) then
30:     if ($E_1.isEqual(E_2)$) then
31:         // Return detected access flaw
32:         return "$Flaw$";
33:     end if
34: end if
35: if ($SSF == True$ && $RESSF == True$ && $ASF == True$ && $CSF == True$) then
36:     if ($!E_1.isEqual(E_2)$) then
37:         // Return detected conflict
38:         return "$Conflict$";
39:     else
40:         // Return detected redundancy
41:         return "$Redundancy$";
42:     end if
43: end if
44: return;
```

Algorithm 2 Policy analyzer (P_1, P_2).

```
1: // Call intra policy analyzer to detect anomalies within policies
2: $intraPolicyAnalyzer(P_1, P_2);$
3: // Call inter policy analyzer to detect anomalies among policies
4: $interPolicyAnalyzer(P_1, P_2);$
```

6.1. Rule-level analysis

The rule analysis algorithm is illustrated in Algorithm 1. It takes two rules $R1$ and $R2$ as input, extract their targets, conditions and effects (Line 1 to Line 19) and compare them to determine if there exists any anomaly between the rules (Line 20 to Line 28). If the target (i.e., subject, resource and action sets) of rule $R2$ is subset of or matches the target of rule $R2$, the condition of $R2$ is a subset of or matches the one of $R1$ (Line 29), both rules have the same effect (Line 30) and $R1$ has precedence order over $R2$, then

Algorithm 3 Intra-policy analyzer (P_1, P_2).

```
1:  // Start analysis
2:  // Detect anomalies within a policy
3:  for i = 1 to 2 do
4:      // Get number of rules in the policy
5:      $NRP_i := P_i.getNumberofRules();$
6:      for $j = 1$ to $NPR_i$ do
7:          for $k = 2$ to $NPR_i - 1$ do
8:              if $(ruleAnalyzer(R_{ij}, R_{ik}).equals(\text{“}Flaw\text{”}))$ then
9:                  // Add both rules to the access flaws set
10:                 $AFSet.add(R_{ij});$
11:                 $AFSet.add(R_{ij});$
12:             end if
13:             if $(ruleAnalyzer(R_{ij}, R_{ik}).equals(\text{“}Conflict\text{”}))$ then
14:                 // Add both rules to the conflicts set
15:                 $CSet.add(R_{ij});$
16:                 $CSet.add(R_{ij});$
17:             end if
18:             if $(ruleAnalyzer(R_{ij}, R_{ik}).equals(\text{“}Redundancy\text{”}))$ then
19:                 // Add both rules to the Redundancies set
20:                 $RSet.add(R_{ij});$
21:                 $RSet.add(R_{ij});$
22:             end if
23:         end for
24:     end for
25: end for
26: return;
```

there is access control flaw between the rules $R1$ and $R2$ (Line 32). If the target of $R2$ intersects with the one of $R1$ (Line 35) and rules $R1$ and $R2$ have opposite effects (Line 36), then $R1$ and $R2$ are conflicting rules (Line 38). Otherwise, if $R1$ and $R2$ have the same effect (Line 39) then the rules are redundant (Line 41). The rule analysis algorithm returns the response to the second-level analysis. In case no anomalies found between the rules, the algorithm returns and empty set.

6.2. Policy-level analysis

The policy analysis algorithm is presented in Algorithm 2. It calls Algorithm 3 to detect anomalies within each policy and Algorithm 4 to detect anomalies among combined policies. It takes two policies $P1$ and $P2$ as input and produces a set of all access flaws FS, conflicts CS and redundancies RS found. In both intra- and inter-policy analyses, the returned responses from the rule analysis calls are appended to the proper sets. For each policy, Algorithm 3 invokes the rule analyzer to check if there exists any flaw, conflict or redundancy between its rules, while inter-policy analyzer presented in Algorithm 4 checks anomalies between rules from different policies taking into account the policies targets and rule combining algorithms.

6.3. Policy set-level analysis

The policy set analysis process is presented in Algorithm 5. It takes a policy set PS as input and reports all access flaws, conflicts and redundancies found between policies and rules. It initializes global set FS, CS and RS and calls the policy analyzer in Algorithm 2 for checking and appending flaws, conflicts and redundancies at both policy and rule levels, within each policy and among different policies.

7. Enforcement

In independent Web service, policies are enforced before invoking its services and they can be evaluated by calling our policy evaluator engine to manage access to the requested resource. In more complex systems like Web services composition [29], policies are enforced and evaluated at the composition level. Therefore, we devise in this work a policy enforcer module, which extracts automatically the resources declared in the generated policies sets and creates calls for the evaluation engine to control access decisions. The calls are built with AspectBPEL constructs, which is an aspect-oriented language that we presented previously [1] to allow modular specification and dynamic integration of new aspects in business processes. The enforcer uses this language to create the evaluation aspects responsible of calling the evaluation engine and weave them in the composition, allowing the decision-making engine to take access control over the resources designated in the policies models.

Algorithm 4 Inter-policy analyzer (P_1, P_2).

```
1:  // Extract elements for analysis
2:  Get targets from both policies
3:  $TAR_1 := P_1.getTarget();$
4:  $TAR_2 := P_2.getTarget();$
5:  $S_1 := TAR_1.getSubjects();$
6:  $S_2 := TAR_2.getSubjects();$
7:  $RES_1 := TAR_1.getResources();$
8:  $RES_2 := TAR_2.getResources();$
9:  $A_1 := TAR_1.getActions();$
10: $A_2 := TAR_2.getActions();$
11: Get combining algorithms
12: $RCA_1 := P_1.getCombiningAlgo();$
13: $RCA_2 := P_2.getCombiningAlgo();$
14: // Start analysis
15: // Detect anomalies among policies
16: $SSF := S_1.\text{intersctWith}(S_2);$
17: $RESSF := RES_1.\text{intersctWith}(RES_2);$
18: $ASF := A_1.\text{intersctWith}(A_2);$
19: if $(RCA_1.equals(RCA_2))$ then
20:     if $(SSF == True$ && $RESSF == True$ && $ASF == True$ && $CSF == True)$ then
21:         $P_1, NRP_1 := P_1.getNumberOfRules();$
22:         $P_2, NRP_2 := P_2.getNumberOfRules();$
23:         for $i = 1$ to $NPR_1$ do
24:             for $j = 1$ to $NPR_2$ do
25:                 if $(ruleAnalyzer(R_i, R_j).equals(\text{“}Flaw\text{”}))$ then
26:                     // Add both rules to the access flaws set
27:                     $AFSet.add(R_i);$
28:                     $AFSet.add(R_j);$
29:                     // Add also both policies
30:                     $AFSet.add(P_1);$
31:                     $AFSet.add(P_2);$
32:                 end if
33:                 if $(ruleAnalyzer(R_{ij}, R_{ik}).equals(\text{“}Conflict\text{”}))$ then
34:                     // Add both rules to the conflicts set
35:                     $CSet.add(R_i);$
36:                     $CSet.add(R_j);$
37:                     // Add also both policies
38:                     $CSet.add(P_1);$
39:                     $CSet.add(P_2);$
40:                 end if
41:                 if $(ruleAnalyzer(R_{ij}, R_{ik}).equals(\text{“}Redundancy\text{”}))$ then
42:                     // Add both rules to the Redundancies set
43:                     $RSet.add(R_i);$
44:                     $RSet.add(R_j);$
45:                     // Add also both policies
46:                     $RSet.add(P_1);$
47:                     $RSet.add(P_2);$
48:                 end if
49:             end for
50:         end for
51:     end if
52: end if
53: return
```

Hereafter we recall the main constructs of the language to help understanding the generated code of the aspects, which are presented further in the text. The `BPEL_Aspect` represents the aspect module, which has a unique identifier. The aspect code starts with `BPEL_Insertion_Point` (i.e., before, after or around) to specify the point where the aspect will be injected followed by the `BPEL_Location_Identifier` that identifies the exact joint point or sets of joint points in the process where the aspect will be activated. In this case, identifies the relevant service. Finally, wrapped between `BeginBehavior` and

Algorithm 5 Policy set analyzer (*PS*).

```
// Initialize anomalies sets
$CSet := \{\}, RSet := \{\}, AFSet := \{\}$
// Get number of policies
$NP := PS.getNumberOfPolicies()$
// Start analysis
for $i = 1$ to $NP - 1$ do
    for $j = 1 + 1$ to $NP$ do
        $anomalies = policyAnalyzer(P_i, P_j)$
    end for
end for
```

```
[1] Aspect FinancialData
[2] ...
[3] BeginAspect
[4]  Before
[5]  <bpel:invoke name="Invoke" partnerLink="WS"
[6]   operation="getFinancialData" inputVariable="Request"
[7]   outputVariable="Response">
[8]  </bpel:invoke>
[9] BeginBehavior
[10] <bpel:invoke name="Invoke" partnerLink="SBEvaluator"
[11] operation="policySetAnalyzer" inputVariable="SBRequest"
[12] outputVariable="SBResponse">
[13] </bpel:invoke>
[14]EndBehavior
[15]EndAspect
```

Fig. 4. Generated aspect.

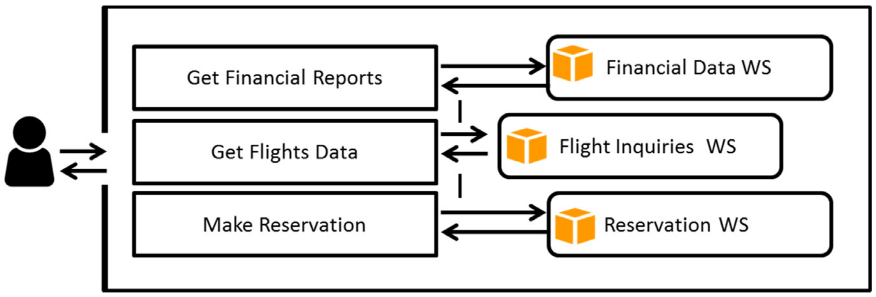

Fig. 5. Flight System (FS).

`EndBehavior`, the `BPEL_Behavior_Code` contains the new behavior code that will be weaved in the composition. In our case, this element represents the call for the evaluation engine.

Taking the same sets generated in Fig. 3, the enforcer generates the corresponding aspect illustrated in Fig. 4. The sets declare the financial data as a resource subject to access control evaluation. Accordingly the enforcer generates FinancialData aspect that calls the evaluation engine (Line 9 to Line 14) before (Line 4) invoking the getFinancialData service (Line 5 to Line 8) in order to check the access right of the requestor.

8. Case study and experiments

To better illustrate our approach, we suggest a Flight System (FS) as a running example (Fig. 5). The system is composed of three partners Web services. The Financial Data WS offers access to financial reports. The Flight Inquiries WS displays the flights with their schedules, available seats and comparable tickets prices, according to the user preferences. Finally, the Reservation WS offers the ability to book flight tickets. The system imposes many policies and rules that govern its services. Considering a policy set *PS*1 for FS that consists of two policies *P*1 and *P*2 and has a permit-overrides combining algorithm. *P*1 defines two rules *R*1 and *R*2. *R*1 permits anyone to access the financial data, while *R*2 is more restrictive. It gives only the admin the permission to access this resource. On the other hand, *P*2 defines two other rules *R*3 and *R*4. *R*3 allows anyone to make reservation in the flight agency system while *R*4 prevents anyone from making reservation after 23:30.

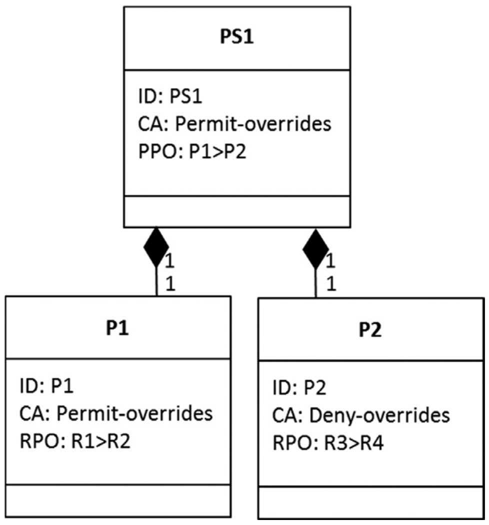

Fig. 6. Policy set *PS*1.

8.1. FS policies specification

Defining the described policies and rules in XML-based format is without doubt verbose, time consuming and error prone. A design level specification is needed since it allows the designer to easily modify the policies in case anomalies are detected within its rules. Whereas in case the modification of policies is not feasible, and there exist anomalies between policies from different parties, such model-driven approach offers the ability to select others participating in composed model that do not cause anomalies when combined together. Following our approach, the user can create a simple UML model that contains: Policy set *PS*1, Policies *P*1 and *P*2, Rules *R*1, *R*2, *R*3 and *R*4, Conditions *C*1 and *C*2, and Targets and their sub-elements (i.e., subject, resource and action) for each rule. The user then applies systematic transformation on the model based on the proposed UML profile. This is done by:

1. Applying `PolicySet` stereotype on `PS1` and specifying its tagged values `ID`, `CA` and `PPO`.
2. Applying `Policy` stereotype on `P1` and `P2` and specifying their tagged values `ID`, `CA` and `RPO`.
3. Applying `Rule` stereotype on `R1`, `R2`, `R3` and `R4` and specifying their tagged values `ID` and `RE`.
4. Applying `Condition` stereotype on `C1` and `C2` and specifying the appropriate operations `Operation`.
5. Applying `Target` stereotype and its substereotypes on the relevant targets elements and specifying the relevant tagged values of `Subject`, `Resource`, and `Action`.
6. Associate the elements together.

Fig. 6 depicts the policy set designation while Figs. 7 and 8 illustrate the policies specification *P*1 and *P*2 and their associated rules *R*1, *R*2 and *R*3, *R*4 respectively.

8.2. Set-based representation of FS policies

According to the set-based constructs that we presented in Section 5, the convertor module generates the sets illustrated in Fig. 9 that correspond to the model created in previous step.

8.3. FS policies analysis

The analyzer module takes care of the detecting anomalies existing in the sets based on the analysis technique and algorithms presented in Section 6. The analyzer generates an analysis report to support the designer in locating the problematic rules within and between policies. Fig. 10 presents a synopsis of the generated analysis report. The highlighted parts demonstrate the capability of the proposed algorithms to detect access flaws, conflicts and redundancies between policies and rules in the policy set.

8.4. Anomalies resolution

XACML supports resolving conflicts between policies and rules using the combing algorithms. Yet it does not provide means to address access flaws and redundancies, which require more customized strategies that can assist the designer. To resolve the detected anomalies, designers have different options. The first can be to cut out the rules overruled by others. The rules to

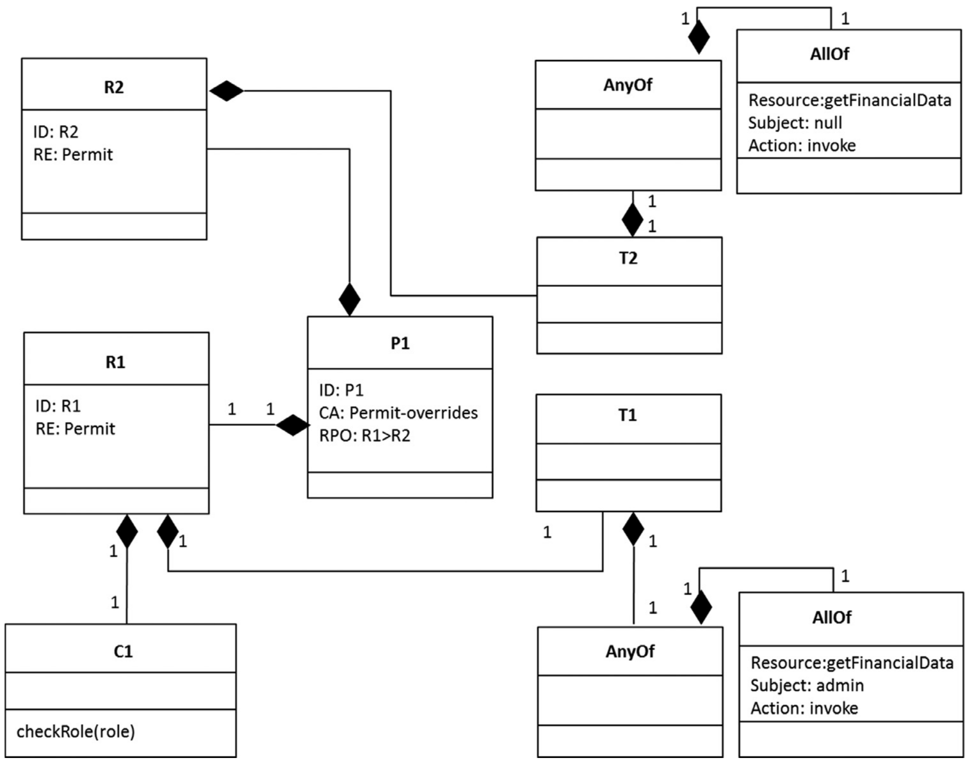

Fig. 7. Policy *P*1 and its associated rules.

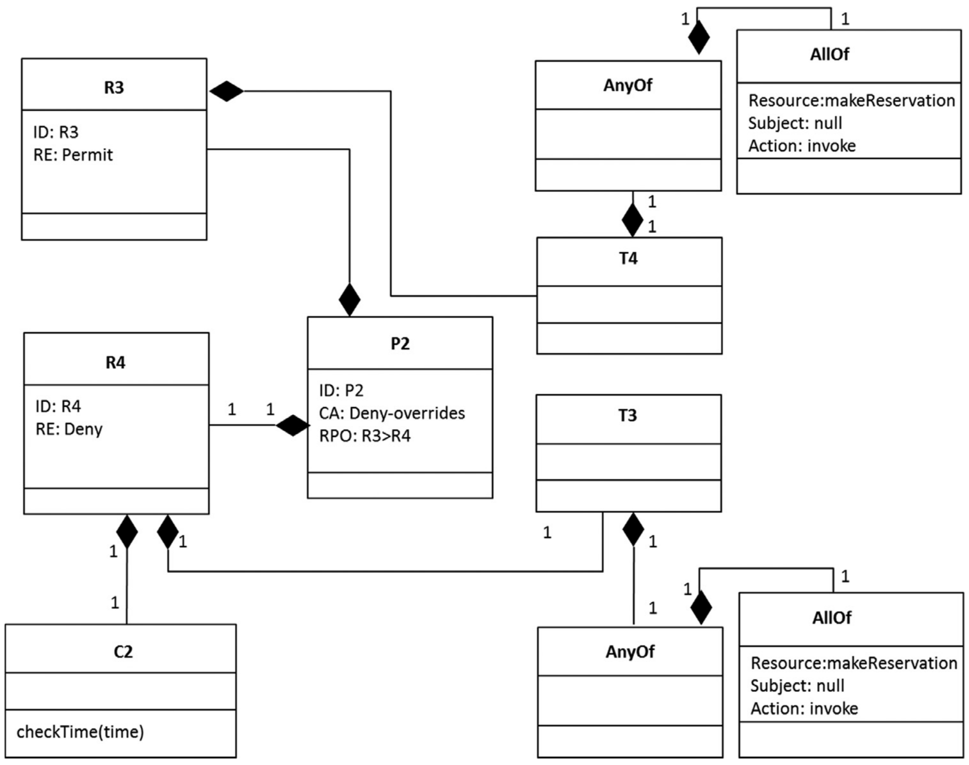

Fig. 8. Policy *P*2 and its associated rules.

```
[1] PS::={PS1,{P1,P2},{P1>P2},{permit-overrides},{},{},{{},{},{}}}
[2] P::={P1,{R1,R2},{R1>R2},{premit-overrides},{},{},{{},{},{}}}
[3] R::={R1, {}, {{},{getFinancialData},{Invoke}},{Permit}}
[4] R::={R2, {{and,{string-equal,{ResourceAttributeDesignator,string,
         getFinancialData}},{string-equal,{SubjectAttributeDesignator,subject-
         id,string,Admin}}},{{Admin},{getFinancialData},{Invoke}},{Permit}}}
[5] P::={P2,{R3,R4},{R3>R4},{deny-overrides},{},{},{{},{},{}}}
[6] R::={R3, {}, {{},{makeReservation},{Invoke}},{Permit}}
[7] R::={R4, {{and,{string-equal,{ResourceAttributeDesignator,string,
         getFinancialData}},{time-greater-than-or-equal,{TimeAttributeDesignator,
         current-time,time,23:30:00}}},{{},{makeReservation},{Invoke}},{Deny}}}
```

Fig. 9. Set-based representation of FS policies model.

```
...
PS1 contains 2 Policies P1 and P2.
    Check Access Flaws in P1.
          …
          R1 and R2 have Equivalent Targets.
          C1 is a Subset of C2.
          R1 and R2 have Same Effect, Permit.
          Access Flaw Detected between R1 and R2.
          FRS = {R1, R2}.
    ...
    Check Redundancies in P1.
          P1 contains 2 Rules R1 and R2.
          ...
          R1 and R2 have Equivalent Targets.
          C1 is a Subset of C2.
          R1 and R2 have Same Effect, Permit.
          Redundancy Detected between R1 and R2.
          RRS = {R1, R2}.
    ...
    Check Conflicts in P2.
          P2 contains 2 Rules R3 and R4.
          ...
          R3 and R4 have Equivalent Targets.
          C4 is a Subset of C3.
          R3 and R4 have Different Effects, Permit and Deny.
          Conflict Detected between R3 and R4.
          CRS = {R3, R4}.
    ...
    Check Conflicts between P1 and P2.
          P1 and P2 have Different Targets.
          No Conflict between P1 and P2.
```

Fig. 10. Synopsis of the detection report.

be eliminated are those which are more generic, while those more restrictive are to be kept in the model. For instance, in our example, eliminating *R*1 from *P*1 would resolve both access flaw and redundancies. *R*2 is more restrictive and hence it would be more secure to keep it in the model. Also removing *R*3 would resolve the conflict (*R*4 is more restrictive). Therefore, a possible extension to this work is to automate these strategies to include them in our proposition.

8.5. FS access control evaluation aspects

After anomalies being resolved, the enforcer generates the corresponding aspects. The latter call the evaluation engine before invoking the requested service in case it is subject to access control checking (i.e., specified in the model).

According to the policies model of FS, access should be controlled over financial data, where access to this resource is restricted to the admin, as well as before making reservation in order to verify the access request time. Figs. 11 and 12 show the generated aspects accordingly. The enforcer weaves these aspects in the composition in order to call the decision-making engine before granting any access to these services.

8.6. Experimental results

In this section, we present the experiments conducted in order to explore the detection rate, scalability and performance of the anomalies analysis process.

The experiments are done on small and large policies that include rules ranging between 400 and 4000. We use 5 policies to vary the number of rules between 400 and 2000, and 10 policies for rules ranging between 2400 and 4000. Conflicted, redundant

```
Aspect FinancialData
...
BeginAspect
 Before
 <bpel:invoke name="Invoke" partnerLink="FinancialDataWS"
  operation="getFinancialData" inputVariable="Request"
  outputVariable="Response">
 </bpel:invoke>
BeginBehavior
<bpel:invoke name="Invoke" partnerLink="SBEvaluator"
operation="policySetAnalyzer" inputVariable="SBRequest"
outputVariable="SBResponse">
</bpel:invoke>
EndBehavior
EndAspect
```

Fig. 11. Aspect for access control over financial data.

```
Aspect Reservation
...
BeginAspect
 Before
 <bpel:invoke name="Invoke" partnerLink="ReservationWS"
  operation="makeReservation" inputVariable="Request"
  outputVariable="Response">
 </bpel:invoke>
BeginBehavior
<bpel:invoke name="Invoke" partnerLink="SBEvaluator"
operation="policySetAnalyzer" inputVariable="SBRequest"
outputVariable="SBResponse">
</bpel:invoke>
EndBehavior
EndAspect
```

Fig. 12. Aspect for access control upon making reservation.

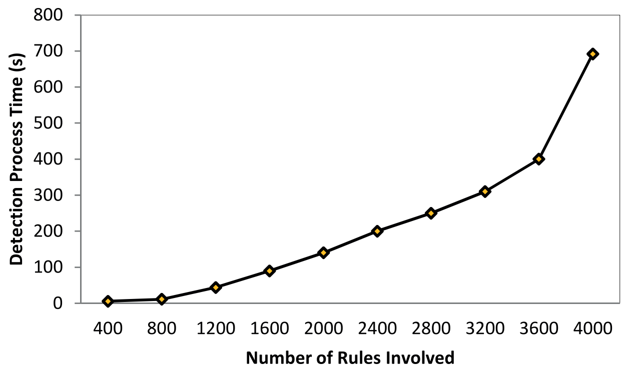

Fig. 13. Detection process performance.

and flawed rules were injected with rate that ranges between 1 and 5 for every 10 rules. An interesting observation in this set of experiments is that the analysis process could always achieve 100% detection rate. As for the processing time, it is calculated based on the average of 100k run for every policy set. The injected anomalies are 10%, and the processing time measured includes the conversion to the set-based representation. Fig. 13 shows the processing time where 1 out of 10 rules/policies cause access anomalies. With 400 rules, it takes 5.54 s to complete the analysis, while it takes 44 s for 1200 rules and 692 s for 4000 rules. These results demonstrate the performance efficiency of the analysis process for reasonable size of policies. Whereas the additional overhead in the other cases is not critical since such analysis is to be performed offline in most cases.

9. Conclusion

This paper addresses the complex specification of XACML policies, which is verbose, time consuming and error prone especially when multiple policies have to be defined to govern access in Web services composition. It tackles also the detection of anomalies within and among policies, at design-level. The proposed approach includes a UML profile to offer model-driven specification of XACML policies. It differs from the literature by its full coverage of all the constructs in the latest XACML version, offering the ability to designate any policy that can be expressed with this language. The proposition includes also a set-based design-level analysis technique that can detect all three anomalies of conflicts, redundancies and access flaws in the policies model. Differently from existing approaches, the proposed technique is based on logical deductions that explore the meaning of the rules declared in the policies and does not make any assumption or exclude any element of the standard XACML language. The real life case study and experimental results presented in this paper prove the efficiency of our approach and demonstrate the capability of the analysis technique showing 100% detection rate with acceptable overhead.

Two possibilities to extend this work in the future. The first proposition is to assist the designer to resolve the detected anomalies by developing and integrating relevant possible strategies compatible with each of them. The second direction is to work on detecting other type of anomalies, which can be influenced by the combined decisions of several partner Web services in the composition.

Acknowledgments

This work has been supported by the Associated Research Unit of the National Council for Scientific Research, CNRS-Lebanon, Lebanese American University (LAU), Khalifa University of Science, Technology & Research (KUSTAR), and NSERC Canada.

References

[1] Tout H, Mourad A, Talhi C, Otrok H. AOMD approach for context-adaptable and conflict-free web services composition. Comput Electr Eng 2015.
[2] Pardal ML, Harrison M, Marques J, et al. Performance assessment of XACML authorizations for supply chain traceability web services. In: Proceedings of the 2012 fourth international conference on computational aspects of social networks (CASoN). IEEE; 2012. p. 378–83.
[3] Tout H, Mourad A, Otrok H. XRML-RBLicensing approach adapted to the BPEL process of composite web services. Serv Oriented Comput Appl 2013;7(3):217–30.
[4] Rissanen E. Extensible access control markup language (XACML), version 3.0 (committe specification 01). Technical report. OASIS; 2010. http://docs.oasisopen.org/xacml/3.0/xacml-3.0-core-spec-cd-03-en.pdf.
[5] Nasim R, Buchegger S. XACML-based access control for decentralized online social networks. In: Proceedings of the 2014 IEEE/ACM 7th international conference on utility and cloud computing. IEEE Computer Society; 2014. p. 671–6.
[6] Jebbaoui H, Mourad A, Otrok H, Haraty R. Semantics-based approach for detecting flaws, conflicts and redundancies in XACML policies . Comput Electr Eng 2015;44:91–103.
[7] Yahyaoui H, Mourad A, Almulla M, Yao L, Sheng QZ. A synergy between context-aware policies and AOP to achieve highly adaptable web services. Serv Oriented Comput Appl 2012;6(4):379–92.
[8] WSBPELT.C.. Web services business process execution language, version 2.0, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2015 [accessed 01.07.15].
[9] Jin X. Applying model driven architecture approach to model role based access control system [Ph.D. thesis]. University of Ottawa; 2006.
[10] Busch M, Koch N, Masi M, Pugliese R, Tiezzi F. Towards model-driven development of access control policies for web applications. In: Proceedings of the workshop on model-driven security. ACM; 2012. p. 4.
[11] Tout H, Mourad A, Yahyaoui H, Talhi C, Otrok H. Towards a BPEL model-driven approach for web services security. In: Proceedings of the tenth annual international conference on privacy, security and trust (PST). IEEE; 2012. p. 120–7.
[12] Tout H, Talhi C, Kara N, Mourad A. Towards an offloading approach that augments multi-persona performance and viability. In: Proceedings of the 12th annual IEEE consumer communications and networking conference (CCNC). IEEE; 2015. p. 455–60.
[13] OASIS. OMG unified modeling language^{TM} (OMG UML) version 2.5. Technical report. OASIS; 2013. http://www.sce.carleton.ca/courses/sysc-5708/f14/UML2_5-ptc-13-09-05.pdf.
[14] Marouf S, Shehab M, Squicciarini A, Sundareswaran S. Statistics & clustering based framework for efficient xacml policy evaluation. In: Proceedings of the IEEE international symposium on policies for distributed systems and networks, POLICY. IEEE; 2009. p. 118–25.
[15] Mizouni R, Tahar S, Curzon P. Hybrid verification integrating HOL theorem proving with MDG model checking. Microelectron J 2006;37(11):1200–7.
[16] Kolovski V, Hendler J, Parsia B. Analyzing web access control policies. In: Proceedings of the 16th international conference on world wide web. ACM; 2007. p. 677–86.
[17] Rao P, Lin D, Bertino E, Li N, Lobo J. An algebra for fine-grained integration of XACML policies. In: Proceedings of the 14th ACM symposium on access control models and technologies. ACM; 2009. p. 63–72.
[18] Huonder F, Joller JM, Rüschlikon Z. Conflict detection and resolution of XACML policies [Master's thesis]. Rapperswil: University of Applied Sciences; 2010.
[19] Liu AX, Chen F, Hwang J, Xie T. Designing fast and scalable XACML policy evaluation engines. IEEE Trans Comput 2011;60(12):1802–17.
[20] Pina Ros S, Lischka M, Gómez Mármol F. Graph-based XACML evaluation. In: Proceedings of the 17th ACM symposium on access control models and technologies. ACM; 2012. p. 83–92.
[21] Marouf S, Shehab M, Squicciarini A, Sundareswaran S. Adaptive reordering and clustering-based framework for efficient XACML policy evaluation. IEEE Trans Serv Comput 2011;4(4):300–13.
[22] Mourad A, Jebbaoui H. SBA-XACML: set-based approach providing efficient policy decision process for accessing web services. Expert Syst Appl 2015;42(1):165–78.
[23] Liu AX, Chen F, Hwang J, Xie T. XEngine: a fast and scalable XACML policy evaluation engine. In: Proceedings of the 2008 ACM SIGMETRICS international conference on measurement and modeling of computer systems, SIGMETRICS'08. ACM; 2008. p. 265–76.
[24] Charfi A, Mezini M. AO4BPEL: an aspect-oriented extension to BPEL. World Wide Web 2007;10(3):309–44.
[25] Mourad A, Ayoubi S, Yahyaoui H, Otrok H. A novel aspect-oriented BPEL framework for the dynamic enforcement of web services security. Int J Web Grid Serv 2012;8(4):361–85.
[26] Yahya I, Turki SH, Charfi A, Kallel S, Bouaziz R. An aspect-oriented approach to enforce security properties in business processes. In: Proceedings of international conference on service-oriented computing, ICSOC 2012. Springer; 2013. p. 344–55.
[27] Braem M, Gheysels D. History-based aspect weaving for WS-BPEL using Padus. In: Proceedings of the fifth European Conference on web services, ECOWS'07. IEEE; 2007. p. 159–67.
[28] Li L, Liu D, Bouguettaya A. Semantic based aspect-oriented programming for context-aware web service composition. Inf Syst 2011;36(3):551–64.
[29] Sheng QZ, Qiao X, Vasilakos AV, Szabo C, Bourne S, Xu X. Web services composition: a decade's overview. Inf Sci 2014;280:218–38.

Azzam Mourad received the Ph.D. degree in electrical and computer engineering from Concordia University, Montreal, Canada. He is an associate professor of computer science at the Lebanese American University. His research interests include information security, Web services, vehicular networks, and formal semantics. He is serving as TPC and reviewer of several prestigious conferences and journals.

Hanine Tout received the M.Sc. degree in computer science from the Lebanese American University, Beirut, Lebanon. She is currently working toward the Ph.D. degree in software engineering from ETS, Montreal, Canada. Her research interests includes Web services security, BPEL, Aspect-Oriented Programming, mobile cloud computing, mobile virtualization and optimization.

Chamseddine Talhi received the Ph.D. degree in computer science from Laval University, Quebec, Canada. He is an associate professor in the department of software engineering and IT at ETS, University of Quebec, Montreal, Canada. He is leading a research group that investigates Smartphone and embedded systems security. His research interests include cloud security and secure sharing of embedded systems.

Hadi Otrok received the Ph.D. degree in electrical and computer engineering from Concordia University. He is an associate professor in the department of ECE at Khalifa University. He works on network and computer security, game theory and mechanism design. He chaired several security-related conferences. He is a TPC member of several prestigious conferences and reviewer of several IEEE and Elsevier journals.

Hamdi Yahyaoui received the Ph.D. degree in computer science from Laval University, Quebec city, Canada. He is an associate professor in the computer science department at Kuwait University in Kuwait. His research interests include Service Oriented Computing, data analysis and security.