An Algebra for Integration and Analysis of Ponder2 Policies

Hang Zhao
Department of Computer Science
Columbia University
zhao@cs.columbia.edu

Jorge Lobo
IBM T. J. Watson Research Center
USA
jlobo@us.ibm.com

Steven M. Bellovin
Department of Computer Science
Columbia University
smb@cs.columbia.edu

***Abstract*—Traditional policies often focus on access control requirement and there have been several proposals to define access control policy algebras to handle their compositions. Recently, obligations are increasingly being expressed as part of security policies. However, the compositions and interactions between these two have not yet been studied adequately. In this paper, we propose an algebra capturing both authorization and obligation policies. The algebra consists of two policy constants and six basic operations. It provides language independent mechanisms to manage policies. As a concrete example, we instantiate the algebra for the Ponder2 policy language.**

I. Introduction

Maintaining security policies in dynamic environments such as mobile networks or virtual environments, where devices in the network or entities in the virtual organization share resources, requires mechanisms to compose and integrate security policies. Traditional security policies largely focus on the specification and management of access control requirement [4], [12]. There are methods and tools developed for access control policy analysis and refinement. For example, [3] defined firewall policy anomalies in both centralized and distributed firewalls, and also introduced a set of algorithms to detect such anomalies. A formal model was presented in [7] to capture IPSec and VPN security policies together with a framework for analyzing policy conflicts. [6] described methods to produce compact sequence of firewall rules while achieving consistency and completeness. Prior to this work, we proposed an algebra for firewall policy integration [13]. An algebra for fine-grained integration of XACML policies was introduced in [11]. It is able to support the specification of a wide range of integration constraints.

However the availability of services in many applications often further requires obligation requirements (see for example in [8]). The questions of how to understand the interactions between access control policies and obligation polices, and how to integrate and compose policies to enforce consistency in a policy-based system, have not yet been adequately investigated. Therefore, in this paper, we further extend the algebra defined in [11] by introducing obligation requirement into the model. We demonstrate the integration and analysis of Ponder2 policies using the new algebra.

The rest of the paper is organized as follows. Section II introduces background knowledge concerning the algebra for XACML policies and the Ponder2 language. Section III presents the detailed policy algebra, discusses its properties and expressiveness. Mechanisms for policy integration and analysis are described in Section IV. Finally, Section V concludes the paper.

Research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

II. Background

In this section, we introduce some background knowledge for subsequent discussions. We start with a summary of the XACML policy algebra from [11], then present an overview of the *Ponder2* policy language [5], [10] and preliminary notions of policy semantics adopted in our work.

A. An Algebra for XACML Policies

In [11], an access control policy P defined a function mapping each access request to a value in $\{Y, N, NA\}$, determining whether the request is allowed (Y), denied (N) or not-applicable (NA). Policy functions are captured by a 2-tuple $\langle R_Y^P, R_N^P \rangle$, where R_Y^P is the set of requests that are permitted, R_N^P the ones that are denied; the rest are NA. A Fine-grained Integration Algebra (FIA), $\langle \Sigma, P_Y, P_N, +, \&, \neg, \Pi_{dc} \rangle$, is introduced where Σ is a vocabulary of attributes names and their domains, P_Y and P_N are two policy constants which permits and denies everything respectively, $+$ and $\&$ are two binary operators, and $\neg$ and Π_{dc} are two unary operators. Operators on policies are described as set operations over $\langle R_Y^P, R_N^P \rangle$. The integration of policies may involve multiple operators based on the definition of FIA expressions. Details of the operators can be found in Section III where we make the appropriate extensions to the original definitions to accommodate obligations. Details can be also found in [11].

B. An Overview of Ponder2

The Ponder2 [10] language provides a common means of specifying security policies that map onto various access control implementation mechanisms for firewalls, operating

978-0-7695-3133-5/08 $25.00 © 2008 IEEE
DOI 10.1109/POLICY.2008.42

systems, databases etc. It primairly supports two types of policies: *authorization policies* that defines which actions are permitted under given circumstances and *obligation policies* that define which actions should be performed in response to an event occurring if specific conditions are fulfilled. Ponder2 specifies policies in a *subject-action-target* (SAT) format, in additional to optional fields such as constraints, triggers etc.

Ponder2 provides both *positive authorization* **auth+** and *negative authorization* **auth-**. Only one type of *obligation policy* is specified, stating that a subject is obliged to perform certain action on that target. An *obligation policy* can be enforced only if the corresponding *authorization policy* has been specified in the system. An *event* field specifies the trigger of the obligation. Optional constraints may apply to both types of policies. These constraints are evaluated against the state of the system.

C. Policy Semantics

We take the simple yet powerful notation of policy semantics introduced in [11] and extended to capture both authorization and obligation policies. Our representation can be easily instantiated to support a large variety of policy languages.

DEFINITION 1. A security policy is defined as an evaluation function $\mathcal{P} : \mathcal{ST} \times \mathcal{A} \rightarrow \mathcal{D}$, where $\mathcal{ST}$ is the set of system states, $\mathcal{A}$ represents a finite set of actions and $\mathcal{D}$ denotes the set of decision tuples $\{\langle \mathcal{D}_a, \mathcal{D}_o \rangle\} = \{\langle Y, Y\rangle, \langle Y, NA\rangle, \langle N, NA\rangle, \langle NA, NA\rangle\}$. The function $\mathcal{P}$ takes a system state $st \in \mathcal{ST}$ and an action $a \in \mathcal{A}$ as input, and returns a decision tuple $\langle d_a, d_o\rangle$ determining whether a is authorized and obliged to execute in state st.

As in [11], our semantics supports three-valued policies when evaluating an authorization request. But the obligation to execute an action may exist (Y) or may not (NA).[1] Note that the set of decision tuples does not include $\langle N, Y\rangle$ and $\langle NA, Y\rangle$ to ensure policy consistency since an obligation requirement intuitively implies that the corresponding positive authorization exists in the system. In an implementation, the system states $\mathcal{ST}$ will most likely encode concepts like principals, i.e. the entity that will execute the action, a target, where the action will be executed and any other detail required by the execution of the action. For the evaluation of Ponder2 policies we can capture a state as follows:

DEFINITION 2. Let $\mathcal{S}$ be the set of subjects, $\mathcal{T}$ be the set of targets, $\mathcal{A}$ be the set of actions, $\mathcal{E}$ be the set of event triggers and $\mathcal{C}$ be the set of conditional constraints. We can now define system states as $\mathcal{ST} = \mathcal{E} \times \mathcal{C} \times \mathcal{S} \times \mathcal{T}$. This definition allows a system state to be described as $st = \langle e, c, s, t\rangle$ consisting of an event trigger $e \in \mathcal{E}$, the conditional constraint $c \in \mathcal{C}$, subject $s \in \mathcal{S}$ and target $t \in \mathcal{T}$ in Ponder2.

TABLE I illustrates some sample policies for a conference reviewing system. Policies $\{P_1, P_2, ..., P_5\}$ are written in Ponder2 language, and are transformed into $\{P'_1, P'_2, ..., P'_5\}$.

[1]We do not consider negative obligations in our model. Because negative obligations can be easily transformed into access control requirement like refrain policies, and thus can be enforced directly.

P_1	**auth+** a=/author→p=/paper /paper.read() **when** submit(a,p)=T
P_2	**auth+** r=/reviewer→p=/paper /paper.read() **when** assign(r,p)=T
P_3	**auth+** r=/reviewer→p=/paper /paper.review() **when** assign(r,p)=T
P_4	**auth−** a=/author→p=/paper /paper.read(), /paper.review() **when** submit(a,p)=T
P_5	**on** assign(r,p)=T r=/reviewer p=/paper **do** /reviewer→/paper.review()
P'_1	$P'_1(\langle$submit(a,p)=T, a=/author, p=/paper$\rangle$, read()$) = \langle Y, NA\rangle$
P'_2	$P'_2(\langle$assign(r,p)=T, r=/reviewer, p=/paper$\rangle$, read()$) = \langle Y, NA\rangle$
P'_3	$P'_3(\langle$assign(r,p)=T, r=/reviewer, p=/paper$\rangle$, review()$) = \langle Y, NA\rangle$
P'_4	$P'_4(\langle$submit(a,p)=T, a=/author, p=/paper$\rangle$, {read(), review()}$) = \langle N, NA\rangle$
P'_5	$P'_5(\langle$assign(r,p)=T, r=/reviewer, p=/paper$\rangle$, review()$) = \langle Y, Y\rangle$

TABLE I
SAMPLE POLICIES FOR A CONFERENCE REVIEWING SYSTEM

DEFINITION 3. Our first algebra components are the following two policy constants $\mathbf{P}_+ : \mathcal{ST} \times \mathcal{A} \rightarrow \langle Y, NA\rangle$ and $\mathbf{P}_- : \mathcal{ST} \times \mathcal{A} \rightarrow \langle N, NA\rangle$. More precisely, $\mathbf{P}_+$ specifies that every authorization request will be allowed in any state and no obligation is required in the system; whereas $\mathbf{P}_-$ says that any authorization request is denied therefore no obligation is required.

It is quite common in a policy-managed system to adopt a default authorization policy: every action is allowed or every action is forbidden. Firewall policy is such an example. Ponder2 also allows the policy administrator to specify a default authorization policy, such as `ALL+` or `ALL-`. Normally, default obligation requirements are not specified during system initialization. The two policy constants allow us to represent default authorization requirement in the system.

III. AN ALGEBRA FOR POLICIES

We are now ready to introduce the extended version of the algebra from [11]. We first define the syntax and semantics of the basic algebraic operations and then discuss their properties and expressiveness in details.

A. Basic Algebraic Operations

The algebra consists of the two constant policies P_+ and P_- and six basic algebraic operations: addition (+), intersection (&), subtraction (−), projection (Π), and two negation operations, one for authorizations ($\neg_a$) and another for obligations ($\neg_o$). Let P_1 and P_2 be two policies to be combined, and P_I be the result from combination. For the ease of our discussion, we introduce three binary operators $\oplus, \otimes, \ominus$ in TABLE II. The first column of each matrix are the evaluation results of P_1 with respect to an input (st, a) and the first row are the results of P_2 with respect to the same input. Each entry denotes the effect of integrating P_1 and P_2 using $\oplus, \otimes, \ominus$ respectively. The effect of the two negation operators $\neg_a$ and $\neg_o$ are also described in TABLE II.

Addition$(+)$. Addition of P_1 and P_2 results in an integrated policy P_I equivalent to the union of the two.

$$\begin{aligned} P_I(st, a) &= P_1 + P_2(st, a) \\ &= P_1(st, a) \oplus P_2(st, a) \end{aligned}$$

Intersection$(\&)$. Intersection of P_1 and P_2 results in policy P_I equivalent to the conjunction of the two.

$$\begin{aligned} P_I(st, a) &= P_1 \& P_2(st, a) \\ &= P_1(st, a) \otimes P_2(st, a) \end{aligned}$$

Negation$(\neg_a, \neg_o)$. Negation of policy P returns a policy P_I

$P_1(st,a) \oplus P_2(st,a)$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$
$\langle Y,Y\rangle$	$\langle Y,Y\rangle$	$\langle Y,Y\rangle$	$\langle NA,NA\rangle$	$\langle Y,Y\rangle$
$\langle Y,NA\rangle$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle NA,NA\rangle$	$\langle Y,NA\rangle$
$\langle N,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle N,NA\rangle$	$\langle N,NA\rangle$
$\langle NA,NA\rangle$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$

$P_1(st,a) \otimes P_2(st,a)$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$
$\langle Y,Y\rangle$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$
$\langle Y,NA\rangle$	$\langle Y,NA\rangle$	$\langle Y,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$
$\langle N,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$
$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$

$P_1(st,a) \ominus P_2(st,a)$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$
$\langle Y,Y\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle Y,Y\rangle$
$\langle Y,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle Y,NA\rangle$
$\langle N,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle N,NA\rangle$
$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$

$P(st,a)$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$
$\neg_a P(st,a)$	$\langle N,NA\rangle$	$\langle N,NA\rangle$	$\langle Y,NA\rangle$	$\langle NA,NA\rangle$
$\neg_o P(st,a)$	$\langle Y,NA\rangle$	$\langle Y,Y\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$

TABLE II

POLICY COMBINATION MATRIX FOR OPERATORS $\oplus, \otimes, \ominus, \neg_a, \neg_o$

which permits (denies) all requests denied (permitted) by P. To maintain the completeness property of the algebra, we introduce two negation operators. Operator $\neg_a$ is used to negate the result of evaluating an authorization request. It does not modify the obligation part with the only exception that $\neg_a P_I(st,a) = \langle N,NA\rangle$ if $P(st,a) = \langle Y,Y\rangle$ (since $\langle N,Y\rangle$ is not valid). Similarly, the operator $\neg_o$ is used to negate the evaluation result of an obligation request without touching the authorization part.

$$\begin{aligned} P_I &= \neg P, \text{ such that} \\ P_I(st,a) &= \neg P(st,a), \text{ where } \neg \in \{\neg_a, \neg_o\} \end{aligned}$$

Subtraction$(-)$. Subtracting policy P_2 from P_1 results in the policy P_I which applies only to those requests that P_2 does not apply to. Also the subtraction operation can be expressed in terms of $\{+, \&, \neg_a, \neg_o\}$.

$$\begin{aligned} P_I(st,a) &= P_1 - P_2(st,a) \\ &= P_1(st,a) \ominus P_2(st,a) \\ &= (P_1 + \neg_a P_2)\&(P_1 + \neg_o P_2)(st,a) \end{aligned}$$

Projection(Π). The projection operation is used to extract a portion of a given policy P. Let $c(\mathcal{ST} \times \mathcal{A})$ be a computable subset of $\mathcal{ST} \times \mathcal{A}$. The projection operation restricts the policy P based on the specified evaluation result of request $(st,a) \in c(\mathcal{ST} \times \mathcal{A})$. The usper-index $\langle d_a, d_o\rangle$ is optional. If it does not appear, the second condition in the "if" is ignored.

$$\begin{aligned} P_I &= \Pi^{\langle d_a,d_o\rangle}_{c(\mathcal{ST}\times\mathcal{A})} P, \text{ such that} \\ P_I(st,a) &= \Pi^{\langle d_a,d_o\rangle}_{c(\mathcal{ST}\times\mathcal{A})} P(st,a) \\ &= \begin{cases} \langle d_a, d_o\rangle & \text{if } (st,a) \in c(\mathcal{ST}\times\mathcal{A}) \\ & \text{and } P(st,a) = \langle d_a, d_o\rangle \\ \langle NA,NA\rangle & \text{otherwise} \end{cases} \end{aligned}$$

So far we have defined the policy algebra consisting of two policy constants and six basic operations. Note that the integration of policies may involve a sequence of operations as discussed in [11].

B. Properties and Expressiveness of the Algebra

The policy algebra has all the algebraic properties listed in [11] except for those involving negation operations. Instead, we have the following properties:

1) $\neg_a(P_1 + P_2) = \neg_a P_1 + \neg_a P_2$;
2) $\neg_a(P_1 \& P_2) = \neg_a P_1 \& \neg_a P_2$;
3) $P_+ = \neg_a P_-$, $P_- = \neg_a P_+$;
4) $\neg_o \neg_o P = P$.

In section II, we introduced two constant policies P_+ and P_-. The other two constant policies $P_Y : \mathcal{ST} \times \mathcal{A} \to \langle Y,Y\rangle$ and $P_{NA} : \mathcal{ST} \times \mathcal{A} \to \langle NA,NA\rangle$ can be produced using P_+ and P_-. For example, $P_Y = \neg_o P_+$ and $P_{NA} = P_+ + P_- = P_+ \& P_-$.

COROLLARY of THEOREM 3 in [11]. Given any policy integration matrix M, let $M(P_1, P_2)$ be the result of integrating policies P_1 and P_2 using matrix M. We can always find an algebra expression $\mathcal{E}$ such that $\mathcal{E}(P_1, P_2) = M(P_1, P_2)$, where $\mathcal{E}$ consists of $\{P_+, P_-, +, -, \&, \neg_a, \neg_o, \Pi\}$. Therefore, our algebra is complete and the detailed proof is removed for space conservation.

IV. POLICY INTEGRATION AND ANALYSIS

In this section, we describe mechanisms for policy integration and analysis using the algebraic operations defined previously.

A. Policy Integration and Analysis

The integration of policies $P_1, P_2, ..., P_n$ can be easily expressed as $P_I = P_1 + P_2 + ... + P_n$. The result of policy integration P_I permits requests that are permitted by at least one of the policies; It denies requests that are denied by at least one of them. If a request is permitted by one policy but denied by another policy, P_I returns $\langle NA,NA\rangle$ and does not make decision on the request. Sometimes, we may also want to extract desired parts of individual policies and then combine them into an integrated one. We can, for example, take the allows from P_1 and the denies from P_2 using $P_I = \Pi^{\langle Y,d_{o_1}\rangle} P_1 + \Pi^{\langle N,d_{o_2}\rangle} P_2$. As we discussed in SECTION II, it is quite common in a policy-managed system to adopt a default authorization policy: every action is allowed (P_+) or every action is forbidden (P_-). Therefore, any request that is evaluated to $\langle NA,NA\rangle$ by the integrated policy P_I will take the result specified by the default policy P_d. Thus we have $P'_I = P_I + (P_{def} - P_I)$, where $P_{def} \in \{P_+, P_-, \neg_o P_+\}$ depending on the system initialization.

It has been long recognized that merely providing a policy editing tool to ensure correct policy syntax is not sufficient. Policies can interact with each other, often with undesirable effects as pointed by [2]. With the support of policy algebra, we can perform policy analysis upon enforcement.

1) Dominance Check: Dominance check is important because it helps to eliminate redundancies. We say that a policy P is dominated by the already existing policy P' if P does not have any effect when added into the policy set because of the existence of policy P', i.e. there is no resource in the domain whose operation will be affected by P. For example, a policy that permits the creation of usernames only if "length(username) ≥ 4" is dominated by another policy for which says "$4 \leq$ length(username) ≤ 10". Given two policies

P and P', we say that P is dominated by P' if and only if $P + P' = P'$ or $P \& P' = P$.

2) Coverage Check: When specifying policies for management systems, the administrator may want to know if explicit policies have been defined for a certain range of input parameters. The input parameters of our interests would be a certain set of *system states* $\mathcal{ST}$, and action set $\mathcal{A}$. That is, given a computable subset $c(\mathcal{ST} \times \mathcal{A})$ of $\mathcal{ST} \times \mathcal{A}$, we want to make sure that $\Pi_{c(\mathcal{ST} \times \mathcal{A})} P(st, a) \neq \langle NA, NA \rangle$ for all input requests $(st, a) \in c(\mathcal{ST} \times \mathcal{A})$.

3) Conflict Detection and Resolution: Policy conflicts can arise due to omissions, errors or conflicting requirement of the administrators specifying the policies. For example, there may be two authorization policies which permit and forbid the same activity; or an obligation policy may define an activity which is forbidden by a negative authorization policy. We say that two policies are in conflict if they cannot be satisfied simultaneously [9]. Given two policies P_1 and P_2, we know that they conflict with each other only if $P_1(st, a) \neq P_2(st, a)$ for some $(st, a) \in \mathcal{ST} \times \mathcal{A}$. A common mechanism to resolve conflicts is to assign proper precedence to policies. Three possibilities are:

- $P_{I_1} = P_1 + (P_2 - P_1)$: gives higher precedence to P_1. That is, $P_I(st, a) = P_1(st, a)$ for all (st, a) such that $P_1(st, a) \neq P_2(st, a)$.
- $P_{I_2} = P_2 + (P_1 - P_2)$: gives higher precedence to P_2. Similarly, $P_I(st, a) = P_2(st, a)$ for all (st, a) such that $P_1(st, a) \neq P_2(st, a)$.
- $P_{I_d} = (P_1 + P_2) + (P_{def} - (P_1 + P_2))$ solves the conflicts by applying the default policy to requests that receive conflicting results from P_1 and P_2.

These three cases can be easily extended to a collection of policies.

B. Examples in Ponder2

In this subsection, we provide some concrete examples of Ponder2 policy analysis. TABLE I lists a few sample policies $\{P_1, P_2, ..., P_5\}$ written in Ponder2 language. They are transformed into $\{P'_1, P'_2, ..., P'_5\}$ using the proposed policy semantics.

EXAMPLE 1: As an example of policy combination, we can perform $P_I = P'_1 + P'_2$ to obtain the integrated policy P_I. More precisely, for a request (st, a), we have $P_I(st, a) = \langle Y, NA \rangle$ if and only if $st \in \{\langle$submit(a,p)=true, a=/author, p=/paper$\rangle$, $\langle$assign(r,p)=T, r=/reviewer, p=/paper$\rangle\}$ and $a =$ read(); Otherwise, $P_I(st, a) = \langle NA, NA \rangle$. That is, the integrated policy P_I allows both the author and the assigned reviewer to read the paper. Similarly, we could perform $P'_I = P'_2 + P'_3$ and the resulting policy P'_I allows the reviewer to read and review the assigned paper.

EXAMPLE 2: By combining P'_3 and P'_5, we observe that $P'_I = P'_3 + P'_5 = P'_5$. This is because in the proposed policy semantics, the enforcement of an obligation policy implies that the corresponding authorization requirement must be specified to ensure consistency.

EXAMPLE 3: We notice that P'_1 and P'_4 conflict with each other. Because $P'_1(st, a) = \langle Y, NA \rangle$ but $P'_4(st, a) = \langle N, NA \rangle$ when $st \in \{\langle$submit(a,p)=T, a=/author, p=/paper$\rangle\}$ and $a =$ read(). We can solve this conflict by giving precedence to P'_1. That is, $P'_I = P'_1 + (P'_2 - P'_1)$, such that $P'_I(st, read()) = \langle Y, NA \rangle$ and $P'_I(st, review()) = \langle N, NA \rangle$. Thus, we allow the author to read his own paper but he cannot review it. In reality, it is the system administrator's decision to assign precedence among conflicting policies.

V. Conclusion

To address the importance of obligation policies, and study the interaction between authorization and obligation requirements, we propose an algebra for Ponder2 policies, and provide mechanisms for policy integration and analysis using the algebraic operations. For future work, we would like to accomplish the algebra implementation using Ponder2 language. We also plan to study the interaction between access control policies and obligation policies in great depth. For example, the current version of the algebra is not able to handle obligations that may depend on more than one system state. We would like to improve it in the future.

Acknowledgments

We would like to thank the Policy 2008 reviewers and Ehab Al-Shaer for their helpful comments and suggestions on improving this paper.

References

[1] "Extensible access control makeup language (xacml) version 2.0," *OASIS Standard*, 2005.

[2] D. Agrawal, J. Giles, K. won Lee, and J. Lobo, "Policy ratification," in *Proceedings of IEEE Policy 2005*, 2005.

[3] E. S. Al-Shaer and H. H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *Proceedings of IEEE INFOCOM'04*, 2004.

[4] P. Bonatti, S. D. C. di Vimercati, and P. Samarati, "An algebra for composing access control policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 1, pp. 1–35, Feb. 2002.

[5] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," in *Proceedings of IEEE Policy 2001*, 2001.

[6] M. G. Gouda and X.-Y. A. Liu, "Firewall design: Consistency, completeness, and compactness," in *The 24th IEEE Int. Conference on Distributed Computing Systems (ICDCS'04)*, 2004.

[7] H. H. Hamed, E. S. Al-Shaer, and W. Marrero, "Modeling and verification of ipsec and vpn security policies," in *Proceedings of IEEE ICNP'2005*, Nov. 2005.

[8] K. Irwin, T. Yu, and W. H. Winsborough, "On the modeling and analysis of obligations," in *ACM Conference on Computer and Communications Security (CCS)*, 2006.

[9] E. C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," in *IEEE Transactions on Software Engineering*, vol. 25, no. 6, 1999, pp. 852–869.

[10] E. Lupu, N. Dulay, A. S. Filho, S. Keoh, M. Sloman, and K. Twidle, "Amuse: Autonomic management of ubiquitous e-health systems," in *Concurrency and Computation: Practice and Experience, wiley*, 2007.

[11] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo, "An algebra for fine-grained integration of xacml policies," *CERIAS Tech Report 2001-124, Purdue University*, 2007.

[12] D. Wijesekera and S. Jajodia, "A propositional policy algebra for access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 2, pp. 286–325, May 2003.

[13] H. Zhao and S. M. Bellovin, "Policy algebras for hybrid firewalls," in *Annual Conference of ITA (ACITA) 2007*, 2007.