Semantic-based Obligation for Context-Based Access Control

Mouiad AL-Wahah, Ahmed Saaudi and Csilla Farkas
College of Engineering and Computing, University of South Carolina, 301 Main St, Columbia, SC, U.S.A.

Keywords: Description Logic, Access Control, OWL, Authorization, Obligation, Policy Rules.

Abstract: In this paper, we present a dynamic and extensible semantic-based obligation framework. Our framework is meant to be used in conjunction with context-based authorization. Our approach is suitable to incorporate dynamically changing obligation requirements. We express obligation requirements and contextual information as ontologies. We employ Description logic and Logic Programming technologies for modeling contexts, privileges and obligations. We show how semantic-based techniques can be used to support adaptive and dynamic obligation for Context-Based Access Control (CBAC) policies. We also show that our framework is expressive enough to incorporate obligation's needs in dynamic environments. Furthermore, we have developed a proof of concept implementation to demonstrate our work.

1 INTRODUCTION

Context-Based Access Control (CBAC) is suitable for dynamic environments (AL-Wahah and Farkas, 2018b). In such environments, the authorization policy should adapt to its ever-changing environment.Integrating obligations into CBAC control policy allows a system to react quickly to new circumstances and change obligation requirements easily when needed (Elrakaiby et al., 2012). However, only a few of the current authorization models (Fornara, 2011; Pérez et al., 2009; Robinson and Purao, 2009; Rubab et al., 2014) provide an obligation for context-based access control policies. Moreover, they provide only limited support for the following requirements:

- Adaptability: Context-Based Access Control (CBAC) should be adaptable to the ever-changing conditions of dynamic environments.
- Extensibility: Executing obligations by users or systems highly depends on application and policy requirements. The design must be extensible to support new types of obligations.
- Modularity: Obligation framework shows support flexible activation and deactivation of obligations.

Our framework provides a dynamic and adaptive semantic-based obligation support for CBAC policies. Dynamic adaptability is provided through policy adjustment operations that keep the policy rules updated with the latest context changes. Obligation are modified: 1) When the context of the requester, resource, or environment changes in a way, that change should be reflected in the access control policy; 2) When an obligation evolves over time. Our approach can be adopted by existing CBAC systems to provide obligation services. Our semantic-based obligation supports capabilities such as checking the access control and obligation policies for conflict and consistency, explaining inferences and helping to instantiate and validate the variables in dynamic environments.

The rest of this paper is organized as follows: In section 2, we present the context-based access control system modeling. Section 3 presents our semantic-based context obligation. Section 4 shows how to check obligation states, and Section 5 contains our implementation results. Related works are surveyed in Section 6, and we conclude with suggestions for future work in Section 7.

2 CBAC SYSTEM MODELING

"Context" has been defined by Dey et al. (Dey et al., 2001) as "any information that is useful for characterizing the state or the activity of an entity or the world in which this entity operates." In CBAC, the system administrator (or resource owner) specifies a set of contexts and defines for each context a set of applicable permissions. When an entity (a user) operates under a certain context, (s)he acquires the set of permissions (if any) that are associated with the active context. When the active context changes, the previ-

Al-Wahah, M., Saaudi, A. and Farkas, C.
Semantic-based Obligation for Context-Based Access Control.
DOI: 10.5220/0008120105350540
In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE 2019)*, pages 535-540
ISBN: 978-989-758-378-0

ous permissions are revoked, and new permissions are assigned (Corradi et al., 2004). Hence, context plays the role of a bridge between the requester and access permissions.

We present a motivating scenario that will be used as a running example throughout the rest of this paper.

Motivating Example *"University Travel Authorization (TA) forms are used to specify permitted travel expenses. Upon approval of a TA form, the traveler and the travel office must oblige to some requirements. For example, a traveler must be on campus when applying for TA, and the traveler is obligated to turn in itemized receipts within fifteen days of the conference. When a traveler submits itemized receipts, the system is obliged to reimburse the traveler."*

We develop a framework to support context-based obligation, see Figure 1.

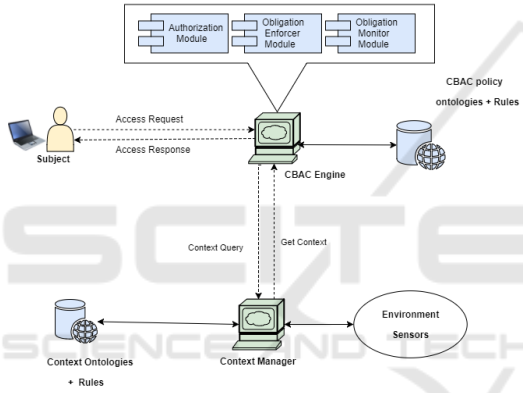

Figure 1: Architectural view of the obligation framework.

In this figure, the CBAC engine is used to enforce the context-based authorization and the obligation policies. CBAC engine supports two modules: obligation enforcer and obligation monitor. Obligation monitor continuously monitors the system state over time. Obligation enforcer is used to create, activate, enforce, and deactivate obligations. The context manager receives requests from the policy engine, asking for a specific context. In its response to these requests, the context manager gathers, annotates, processes, and reasons over the gathered data to produce the context. Then it sends the resulted context to the CBAC policy engine. System ontologies are used for semantic-based system description. This enables the representation of both fine-grained and coarse-grained obligation requirements. The requirements are used to derive dynamic obligation policies, which are used later to enforce and dispense the privileges and commitments of the obligation.

The obligation ontology, OBL, is shown in Figure 2. Once an *Obligation* is created, it can be in one of two temporal states: *Persistent* or *Transient*. In case of persistent obligation, all the data associated with the obligation should be saved by the system. For transient obligation, the data associated with the obligation is only available temporarily. The created obligation also can be in one of the following operational states, $Fulfilled$, $Pending$, $Violated$ (these are specified by CBAC ontology as we will see later).

Obligation can also be a system obligation or a user obligation. The data property $isUserObl$ allows to represent this property. Value of $True$ means it is a user obligation, while $False$ value is for system obligation. An obligation is imposed on a $Subject$ (a system or a user) to oblige her/him/it to perform some $Action$ before getting further access to a $Resource$ (data or services). Deadlines of the obligations are specified using $hasTFrame$ object property, which in turn uses two properties $hasStartTime$ and $hasEndTime$.

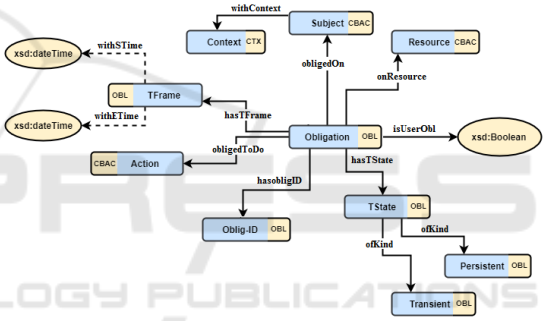

Figure 2: Obligation ontology.

2.1 Context-Based Access Control Model

In this section, we present a brief overview of a CBAC approach suggested by (AL-Wahah and Farkas, 2018a) in which we define Context-Based Access Control model. In this model, access requests are evaluated based on the contexts associated with the subject and the requested resources. Each request is matched with context's metadata that specifies and activates the policy rule to be enforced. The authors use rule-based Logic Programming (LP) to encode context and policy rules.

Definition 1. (Access Control Policy (ACP) Rules). An access control policy rule is a 6-tuple $\langle s, sc, rs, rc, p, ac \rangle$, where $s \in Subject$, $rs \in Resource$, sc, $rc \in Context$, where sc is the subject's context and rc is the resource context (optional), $p \in Permission$=$\{$"$Deny$","$Permit$"$\}$, and $ac \in Action$=$\{read, write, delegate, revoke\}$.

Table 1: Access Authorization rules example.

No.	Rule
Rule:	A student who is with context *AtCampusStudent* can access online TA application.

Each rule is instantiated when an access request is received.

Definition 2. (Access Request (AR)). An Access request is given as a triple $\langle s, rs, ac \rangle$, where $s \in Subject$, $rs \in Resource$, $ac \in Action$.

When an access request is received, the policy engine requests the contexts of the subject *s* and the resource *rs*. Assume the context of *s* is *sc*. If using the context *sc*, the policy engine can derive a permission, i.e., *p* is "*Permit*", and there is no conflict, CBAC grants the access permission for the request. Otherwise, the request is denied.

We represent our model using the OWL-DL ontologies (Hitzler et al., 2010). The context model is built around the concept of contextual attribute. Contextual attributes model the physical/logical environment such as location and temperature. Specific context subclasses can be represented under Generic Concept *Context*. Each subcontext class consists of attribute values and constants.

The *active context* holds the current context of an entity at a specific instant of time. Active context reflects a snapshot of an entity's context. We should note that the context ontology can be extended or shrunk by adding or removing subcontexts or by adding or removing contextual attributes to the subcontexts. From our running example, we build the access control authorization rule as shown in Table 1.

This rule states that if a subject requests an access of the type *TAaccess*, and the subject *s* has a context *AtCampusStudent* then the request is permitted.

3 SEMANTIC-BASED OBLIGATION FOR CBAC

In traditional CBAC authorization, the context plays the role of a bridge between a subject requesting an access and the requested resource. If a subject has the context, then (s)he can access the resource. In obligation-supported CBAC authorization, the bridge is represented by a context plus the required obligation. We differentiate between two types of the context: reference context and obligation context.

(**Reference Context** *c*). A context that is not associated with an obligation and is instantiated by context manager.

(**Obligation Context** *oc*). A context that is dynamically generated by CBAC engine and is associated with an obligation.

By creating a new context of obligation, we avoid modification of the authorization framework and the CBAC policy. Obligation context *oc*, is generated based on reference context. That obligation is always related to a specific action (or a set of actions). Also, the obligation must be fulfilled for its obligation context to be satisfied. An obligation context is dynamically generated by CBAC engine, while a reference context *c* is instantiated by context manager. Creation of specific obligation is achieved in conjunction with an authorization decision enforcement. After its creation, an obligation must be activated. Activation of an obligation is triggered by certain events (such as time elapsing and action applied on a resource).

4 CHECKING OBLIGATION STATES

One of the crucial tasks in managing obligations is how to define and track its states. We propose three major operational states. These are as follows:

- Fulfilled: An obligation is Fulfilled when its conditions are satisfied by the subject.
- Pending: Obligation can be in Pending state in two situations: 1) The action requested by a subject is issued before the start time of the obligation. 2) The action requested by a subject may not relate to the obligation, but the due date/time of the obligation is still valid (does not elapse yet).
- Violated: The action requested by a subject is related to the obligation, but the due date/time of the obligation is invalid (already elapsed)

The CBAC ontology uses three properties to control the obligation process. These properties are *committedTo*, *hasObligation* and *oblig_ID*. The property *committedTo* is used by CBAC engine to check if there is an obligation associated with the current decision. If there is one, it calls the *oblig_enforcer* module to enforce it via executing obligation policy specified for that matter. Property *hasObligation* is used to create an obligation and link it to the current decision so that the subject will further be granted more privileges if (s)he/it satisfies it. Each obligation has a unique identifier called *oblig_ID*. To monitor obligations, the CBAC engine checks the operational states of the obligation. The values of these states are modified by obligation enforcer and being monitored by obligation monitor.

The obligations are specified using DL axioms. For example, the obligation that requires a subject to finalize TA itemized receipts is represented using the DL axiom shown in Figure 3.

$$Oblig \equiv Obligation \sqcap \exists obligedOn.Subject \sqcap \exists hasTState.TState \sqcap \exists withContext.Context \sqcap \exists hasObligID.Oblig\_ID \sqcap \exists onResource.Resource \sqcap \exists hasTFrame.TFrame \sqcap \exists obligedToD.Action \sqcap \exists isUserOblig.\{xsd : Boolean\}$$

Figure 3: Generic obligation axiom.

Using this generic obligation, we will specify another obligation to represent a reference obligation. For example the reference obligation for our example above is given by the DL axiom shown in Figure 4.

$$TAOblig \equiv Obligation \sqcap \exists obligedOn.Subject \sqcap \exists hasTState.TState \sqcap \exists withContext.Context \sqcap \exists hasObligID.Oblig\_ID \sqcap \exists onResource.Resource \sqcap \exists hasTFrame.(TFrame [\geq 2019-09-01T08:00:00, \leq 2019-09-014 T17:00:00])$$

Figure 4: Reference DL obligation axiom.

Assume we have the following obligation business rule:

Table 2: Obligation rule for the running example.

No.	Rule
Rule:	A student who is with context *FinalizingTAStudent* can finalize the on-line TA application.

We should note that the authorization decision in presence of obligation may be different than the authorization decision alone. We represent obligation policies in a predicate form as follows:
$is\_obliged(CBAC, p)$: To check if the decision is associated with an obligation to be enforced. The first parameter is CBAC ontology and the second parameter is the decision previously returned by $evaluate(s, sc, r, CBAC, RS)$ function. This function is called after *Permit* decision is made.
$is\_committed(CBAC, p)$: This predicate, if returns *True*, means that the request is associated with an obligation and that obligation must be enforced before granting access. Same as $is\_obliged(CBAC, p)$ function, except that this function is called before the *Permit* decision is made.
$deactivate(OBL, CBAC, oblig\_ID)$: Deactivate the obligation by deallocating its instances from *OBL* and *CBAC* ontologies.
$oblig\_ID \leftarrow get\_oblig\_ID(s, CBAC)$: Gets the obligation identifier.
$enforce\_oblig(oblig\_ID, CBAC, OBL, s, sc, ac, r, OblP)$: Enforces the obligation for the request in hand. This predicate reads as follows: subject s working under context sc should take the action ac on the resource r only when (s)he fulfills the obligation which is recognized by an identifier $oblig\_ID$ and parameters $OblP$.

Algorithm 1: Obligation-Supported Access Authorization.

Input: *CBAC*, CBAC ontology, *OBL* is a obligation ontology, *RQ* is an Access Request, *RS* is access control rule-set.
Output: Access decision, either "Deny" or "Permit".

```
1:  RT ← parse(RQ)                                   ▷ RT = ⟨s, r, ac⟩
2:  if RT = access then
3:      sc ← getContext(s);
4:      p ← evaluate(s, sc, r, CBAC, RS);
5:      if p = "Permit" AND noConflict(p, s, sc, r, CBAC, RS) then
6:          if is_obliged(CBAC, p) = True then
7:              oblig_ID ← get_oblig_ID(s, CBAC)
8:              enfocre_oblig(oblig_ID, OBL, Oblig(s, sc, ac, rs, [ts, te], ps, tss))
9:              Permit_Access;
10:             exit();
11:         end if
12:     else
13:         Deny_Access;
14:         exit();
15:     end if
16: else
17:     return("Not access request");
18: end if
```

The function $evaluate(s,sc,rs,CBAC,RS)$, takes the access control ontology CBAC as input and achieves the DL-based reasoning to get the inferred model $infmodel$. The access control policy rules, represented as Jena forward inference rules, will be applied on $infmodel$ to derive an access decision. The function $getrulesnumber(RS)$ returns the number of rules in the access control policy rule-set *RS*. The Boolean function $noConflict(p,s,sc,rs,CBAC,RS)$, works just like $evaluate(s,sc,rs,CBAC,RS)$ function but it returns a Boolean value if two conflicting rules ("*Permit*" and "*Deny*") are fired at the same time.

5 IMPLEMENTATION

Our obligation implementation is carried out on Windows 8.1 machine with 1.9 GHz CPU, 4 GB RAM

and 500 GB HDD. We used the on-line Freedatagenerator to generate unduplicated data instances in the form of excel sheets. Then we use Cellfie plug-in for Protègè to assert these instances to the model ontologies. The rules used for this purpose are based on Java Script Object Notation (JSON) (Friesen, 2019). We use SWRL (Semantic Web Rule Language) to specify and trigger operational state transitions. For situations that need to delete an instance from the knowledge base, we use Java-based API.

The obligation's state checking and transition are implemented using SWRL rules. When executed, these rules transit the obligation operational status from state to state if the conditions of the transition are met. Because obligation express future conditions that may not exist in the present time (the time of a request), we randomly generate events (actions-based or temporal) that trigger obligation policies. These events are generated using our Java application and fed into the model ontologies. After DL-reasoning is complete, the execution of SWRL rules follows. These two steps derive all the deductions needed to make the authorization decision. Figure 5 shows the results of our obligation approach simulation. It displays the time needed for different number of obligations' enforcement. In Figure 6, we display the number of DL axioms required to encode the different number of obligations.

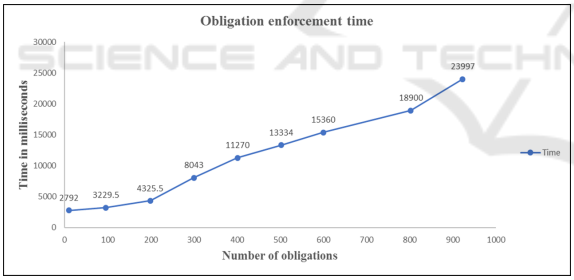

Figure 5: Obligation Enforcement Time for Different Obligation Numbers.

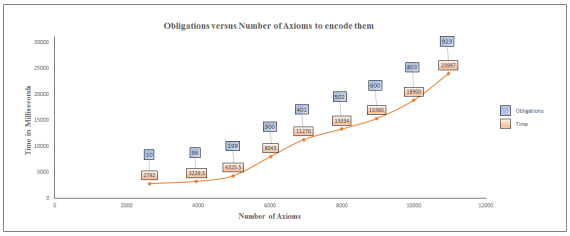

Figure 6: Number of Obligations versus Number of Axioms Processing Time.

6 RELATED WORK

Obligation in access control has been extensively studied by researchers. For example, Hilty et al. (Hilty et al., 2005) propose a formal framework for enforcing obligation policies through using Distributed Temporal Logic (DTL) to classify data protection requirements. Bettini et al. (Bettini et al., 2002) propose an approach for specifying and evaluating provisions and obligations in Access Control (AC). Their method is based on the expression of Datalog Rules and reasoning for evaluating authorization and obligation decisions. They use the term "provision" to refer to the conditions that must be met before data is released after a request is received. We should note that provisions are encoded within our semantic-based authorization model. Authors in (Şensoy et al., 2010) provide Polar, a framework for the semantic definition and enforcement of permission, prohibition and obligation statements. Their definition of an obligation involves activation conditions and contents. Polar uses SPARQL-DL (Pérez et al., 2009) queries for checking the fulfillment of consumer obligations using reasoning, where activation conditions are translated into standard SPARQL queries (Marfia et al., 2015). Chen et al. (Chen et al., 2012) define a model that engages obligations in the environment of risk-aware access control. Ardagna et al. (Ardagna et al., 2010) present an identity management system, called PRIME. PRIME supports autonomous credentials and policy languages that support anonymous credentials. While the policy language supports obligations, it is limited regarding the dynamic instantiation of these obligations. Pontual et al. (Pontual et al., 2011) present an approach to support user obligation enforcement. The authors propose the property of accountability to model unfulfilled obligations. A novel aspect of their work is to consider authorization needs for fulfilling obligations as well as the dependency among multiple obligations. However, it is unclear how the proposed approach accommodate dynamically changing environments.

7 CONCLUSIONS AND FUTURE WORKS

We present a semantic-based formal specification for obligation policies and show how to use obligation policies in conjunction with authorization decision making. A novelty of our work is that we support dynamically generated obligation. This is a crucial requirement in an environment where the context of access requests changes over time.

Obligation is regarded as one of the requirements associated with a specific action. If subjects can delegate their privileges and there is an obligation combined with these privileges, then we need to investigate delegation of obligation as well. Another direction for future work is to engage trust management with authorization and obligation.

REFERENCES

AL-Wahah, M. and Farkas, C. (2018a). Context-aware iot authorization: A dynamic and adaptive approach. In *13th International Conference for Internet Technology and Secured Transactions. (ICITST-2018)*, pages 64–72. Infonomics Society.

AL-Wahah, M. and Farkas, C. (2018b). Context delegation for context-based access control. In *2nd International Workshop on A.I. in Security*, pages 70–79. ECML.

Ardagna, C. A., Camenisch, J., Kohlweiss, M., Leenes, R., Neven, G., Priem, B., Samarati, P., Sommer, D., and Verdicchio, M. (2010). Exploiting cryptography for privacy-enhanced access control: A result of the prime project. *Journal of Computer Security*, 18(1):123–160.

Bettini, C., Jajodia, S., Wang, X., and Wijesekera, D. (2002). Provisions and obligations in policy management and security applications. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 502–513. VLDB Endowment.

Chen, L., Crampton, J., Kollingbaum, M., and Norman, T. (2012). Obligations in risk-aware access control. In *Privacy, Security and Trust (PST), 2012 Tenth Annual International Conference on*, pages 145–152. IEEE.

Corradi, A., Montanari, R., and Tibaldi, D. (2004). Context-based access control management in ubiquitous environments. In *Proceedings of the 3rd IEEE International Symposium on Network Computing and Applications*, pages 253–260. IEEE.

Dey, A., Abowd, D., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16(2):97–166.

Elrakaiby, Y., Cuppens, F., and Cuppens-Boulahia, N. (2012). Formal enforcement and management of obligation policies. *Data & Knowledge Engineering*, 71(1):127–147.

Fornara, N. (2011). Specifying and monitoring obligations in open multiagent systems using semantic web technology. In *Semantic agent systems*, pages 25–45. Springer.

Friesen, J. (2019). *Java XML and JSON, Document Processing for Java SE*. APress.

Hilty, M., Basin, D., and Pretschner, A. (2005). On obligations. In *European Symposium on Research in Computer Security*, pages 98–117. Springer.

Hitzler, P., Krötzsch, M., and Rudolph, S. (2010). *Foundations of Semantic Web Technologies*. Chapman and Hall/CRC Press.

Marfia, F., Fornara, N., and Nguyen, T. (2015). Modeling and enforcing semantic obligations for access control. In *Multi-Agent Systems and Agreement Technologies*, pages 303–317. Springer.

Pérez, J., Arenas, M., and Gutierrez, C. (2009). Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):1–16.

Pontual, M., Chowdhury, O., Winsborough, W. H., Yu, T., and Irwin, K. (2011). On the management of user obligations. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 175–184. ACM.

Robinson, W. and Purao, S. (2009). Specifying and monitoring interactions and commitments in open business processes. *IEEE software*, 26(2):72–79.

Rubab, I., Ali, S., Briand, L., and Traon, Y. L. (2014). Model-based testing of obligations. In *2014 14th International Conference on Quality Software*, pages 1–10. IEEE.

Şensoy, M., Norman, T., Vasconcelos, W., and Sycara, K. (2010). Owl-polar: Semantic policies for agent reasoning. In *International Semantic Web Conference*, pages 679–695. Springer.