An Integrated Approach for the Enforcement of Contextual Permissions and Pre-obligations

Yehia Elrakaiby, Frédéric Cuppens and Nora-Cuppens Boulahia
Institut TELECOM/TELECOM-Bretagne
2 rue de la Châtaigneraie, 35576 Cesson Sévigné Cedex, France
Email: {yehia.elrakaiby, frederic.cuppens, nora.cuppens}@telecom-bretagne.eu

Abstract—Pre-obligations denote actions which may be required before access is granted. The successful fulfillment of pre-obligations authorizes the requested access. Thus, pre-obligations induce interactions between the obligation and authorization policy states. This paper studies these interactions by formalizing the evolution of the authorization and obligation states when pre-obligations are supported. The main advantage of the presented approach is that pre-obligations are given both declarative semantics based on predicate logic and operational semantics based on Event-Condition-Action (ECA) rules. Furthermore, the presented framework enables policy designers to easily choose to evaluate any pre-obligation either (1) statically (an access request is denied if the pre-obligation has not been fulfilled); (2) or dynamically (users are given the possibility to fulfill the pre-obligation after the access request and before access is authorized).

I. INTRODUCTION

Traditionally, security policy systems provided a simple yes/no answer to access requests. With the evolution of information systems, it was recognized that access often depends on some user-actions being performed before access is granted. For instance, we may have an access rule which specifies that users are allowed to download music files provided that they pay 1$ first. In this case, when some user requests to download the latest single of Muse, s/he should be asked to pay 1$. When the payment is made, the user is allowed to download the requested file. Such requirements are called pre-obligations. Neither traditional access control models such as DAC [1] and RBAC [2] nor more recent contextual security models such as ASL [3] and OrBAC [4] support pre-obligations: In these models, an access request is only allowed if the set of conditions associated with the access request are true at the moment of the request.

The support of pre-obligations in the policy language provides several advantages. First, it provides additional expressiveness since it enables policy administrators to specify that subjects may fulfill some of the access requirements after they make their access request. Furthermore, it separates the expression of requirements from the functional specification (the application code) of the system. Thus, the analysis of policy requirements is simplified and administrators are able to modify the behavior of the system by updating policy rules without recoding the application.

To support pre-obligations, a number of works [5], [6], [7] subordinate obligations to access control rules. This approach has some limitations. For instance, obligations are only activated after access requests and general obligations are not supported. In addition, it generally produces intricate access control policies since permissions and obligations are often specified within the same rule. This is the approach used in [7] to specify permissions and their associated pre-obligations. However, the main limitation of previous works on pre-obligations is that none formalized the effects of supporting pre-obligations on the evolution of the authorization and obligation policy states. This is essential to provide a deeper understanding of pre-obligations and their working in information systems. In addition, this formal approach allows the study and the analysis of change in the authorization and obligation policy states in the presence of obligations. Therefore, it enables, for instance, to derive plans to reach some particular authorization states [8], [9] or to explain the deactivation of pre-obligations after permission activation.

In this paper, we study the specification and the enforcement of pre-obligations. In our approach, we formalize the enforcement of pre-obligations using an extension of the language $\mathcal{L}_{\text{active}}$ [10]. $\mathcal{L}_{\text{active}}$ enables the description of change in state using concepts from action specification languages [11]. Thus, it enables the reasoning about state evolution and the study of interactions between pre-obligations and the authorization and obligation policy states. $\mathcal{L}_{\text{active}}$ also supports the specification of reactive behavior in the form of active rules. This feature enables us to provide formal operational semantics for the enforcement of pre-obligations.

To simplify the expression of pre-obligations in access control rules, we specify pre-obligations in the form of context expressions. A security rule context [12] denotes a set of conditions which have to be true for the security rule to be effective. For instance, a context $\text{during\_working\_hours}$ may hold (be true) every working day from 8 in the morning until 6 in the afternoon. In our approach, context rules may be used to specify requirements which state that some user-action should be taken. These contexts are called pre-obligation contexts. We support two evaluations of pre-obligation contexts: The static (traditional) evaluation requires that pre-obligation actions be taken before access
requests are made. The dynamic evaluation, on the other hand, enables the fulfillment of pre-obligation requirements after access requests.

This is an extended version of the paper [13] which appeared in ARES 2010. In particular, we extend our pre-obligation selection algorithm to clarify the formal model and we detail the different aspects of our approach. Furthermore, we consider state contexts in the policy language to simplify policy specification. The remainder of the paper is organized as follows. Section II presents some motivating examples. In Section III, we present our formalization language. Section IV introduces the basic entities of our policy language and shows the description of the application domain. In Section V, we introduce our context language. We then introduce security rules in Section VI. Section VII formalizes policy management and enforcement. In Section IX, we present the enforcement of the policy when pre-obligations are supported. In Section X, we present an application example to discuss the selection and activation of pre-obligations and the evolution of the authorization and obligation policy states when pre-obligations are fulfilled/violated. Finally, Section XI discusses related works and Section XII concludes the paper.

II. Motivating Example

Consider the following access control requirement:

\[ r_1: \text{Cabinet members may contact the president provided that they are authenticated using their personal electronic card} \]

In a traditional access control system, this requirement is enforced as follows: when some subject who is a cabinet member requests to contact the president, the request is authorized if the subject is authenticated using his/her electronic card. Otherwise, the request is denied. This means that the verification of the fulfillment of pre-obligations consists of checking a history of previous action occurrences. This approach is inflexible for the enforcement of \( r_1 \) since it would be more convenient to allow the subject to authenticate after s/he requests to contact the president. Then, when the subject is successfully authenticated, the subject is allowed to contact the president. Thus, when pre-obligations are evaluated dynamically, the system would appear more flexible to the user. Note that this requirement is similar to the following usage requirement enforced by many web sites today “Subjects are allowed to proceed with their download provided that they successfully enter the captcha text printed on the screen”.

To provide such flexibility in the enforcement of access control requirements, we consider that requirements denoting user-actions may be defined as pre-obligations. In this case, when an access request is made, the subject is requested to satisfy the missing pre-obligation requirements (authenticate). When these pre-obligations are fulfilled, the requested access is granted. Figure 1 compares the traditional enforcement of access control policies with their enforcement when pre-obligations are supported.

We now consider this second access control rule:

\[ r_2: \text{The secretary of defense may contact the president provided that s/he is authenticated by password and that s/he is in some secure location} \]

Assume the policy includes both the rules \( r_1 \) and \( r_2 \) and that it is possible to ask the defense secretary to move to a secure location. In this case, when the secretary of defense (who is also a cabinet member) requests to contact the president from an insecure location, two alternative sets of pre-obligations are possible: (1) authenticate by card as specified in \( r_1 \), (2) or authenticate by password and move to a secure location as specified in \( r_2 \). One possible way to deal with this situation is to consider the random selection of one of these two pre-obligation sets. This however clearly represents an unacceptable behavior. Therefore, we choose to allow the association of pre-obligations with weights. For instance, if the pre-obligation to authenticate by electronic card is given a lower weight than the sum of the weights of the two pre-obligations to authenticate by password and to move to a secure location, the pre-obligation with the lower weight (authenticate by card) is selected. On the other hand, if the secretary of defense is in a secure area, s/he is asked to authenticate by password if the pre-obligation to authenticate by password is given less weight than the pre-obligation to authenticate by card. This situation illustrates the importance of the dynamic selection of pre-obligations which takes into account which pre-obligations are and which are not fulfilled at the moment of the access request.

III. The Formalization Language

To formalize the effects of the support of pre-obligations on the authorization and policy states and to enable the study of their properties, we consider the language \( L_{active} \) [10]. \( L_{active} \) enables the description of change in state using concepts from action languages. It also supports the
actions appearing in one of the rules in the triggered rules set to process. Active rules are assigned priorities. Therefore, the action selection function returns the sequence of actions appearing in one of the rules which have the highest priority in the triggered rule set. The state stops evolving after the processing of all the actions in an input sequence if the triggered rule set is empty.

**IV. BASIC ENTITIES & DOMAIN DESCRIPTION**

In this section, we present the basic entities of our policy language and the description of the application domain.

**A. Basic Entities**

We consider a sorted first-order language which includes finite sorts for the entities: subjects $S$, objects $O$, actions $A$ and contexts $C$. Entities may have attributes. For instance, the application dependent Name($s,n$) means that the name of $s$ is $n$. We also consider three relations to enable the specification of security rules for groups of subjects, actions and objects respectively: Subjects are empowered into roles using the relation Empower(Subject, Role), actions, i.e. programs, are considered implementation of some activity using the relation Consider(Action, Activity) and objects are used in views using the relation Use(Object, View). Security rules may be specified using the abstract entities of roles, activities and views as well as using subjects, actions and objects.

**B. Domain Description**

In this paper, we study the evolution of the policy state when change in state occurs. Therefore, we assume that a description of change in state which results from action occurrences in the system is given. Action occurrences represent external events which occur in the environment. We consider that action occurrences of the form $Do(S,A,O)$ denote that subject $S$ has taken the action $A$ on the object $O$. The effect on action occurrences on the state is specified using effect law propositions. For instance, we may specify the effect of the action authenticate by card as follows.

$$\text{Do}(S, \text{authenticate by card}, O)$$

causes $\text{Authenticated}(S, \text{card})$

if $\text{Type}(O, \text{authentication server})$

This effect law specifies that the fluent $\text{Authenticated}(S, \text{card})$ starts to hold (be true) in the state after the action authenticate by card is executed by $S$ on some authentication server $O$. It is reasonable to assume that subjects do not remain authenticated forever. We will therefore assume that the fluent $\text{Authenticated}(S, \text{card})$ seizes to hold when $S$ logs off the system. This is specified as follows.

$$\text{Do}(S, \text{log-off}, \text{system})$$

causes $\neg \text{Authenticated}(S, \text{card})$

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fluents</td>
<td>Facts describing the system state.</td>
<td></td>
</tr>
<tr>
<td>Actions</td>
<td>Possible actions in the system. Action occurrences update the fluent state by adding or removing fluents to or from the state.</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>Define changes in state, generally due to action execution, at which policy updates are needed.</td>
<td></td>
</tr>
<tr>
<td>Rule Names</td>
<td>$ECA$ rule identifiers. An $ECA$ rule states that when event occurs and if conditions are true, then actions are executed. ($ECA$) rules (also called active rules) update the applied policy when particular events are detected.</td>
<td></td>
</tr>
</tbody>
</table>

**Table I**

$\mathcal{L}_{\text{active}}$ SORTS

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effect Law</td>
<td>$a$ causes $\neg f$ \textbf{if} $p_1,\ldots,p_n$ \textbf{then} $f$ true in the state next state.</td>
<td></td>
</tr>
<tr>
<td>Event Definition</td>
<td>$e$ after $a$ \textbf{if} $p_1,\ldots,p_n$ \textbf{then} $e$ true in the state next state.</td>
<td></td>
</tr>
<tr>
<td>Active Rule</td>
<td>$r : e$ \textbf{initiates} $[\alpha]$ \textbf{if} $p_1,\ldots,p_n$ \textbf{then} $e$ true in the state next state.</td>
<td></td>
</tr>
</tbody>
</table>

$a$: Action Symbol  
$f, p_1, \ldots, p_n$: Fluent Literals  
$e$: Event Symbol  
$r$: Active Rule Symbol

**Table II**

$\mathcal{L}_{\text{active}}$ PROPOSITIONS

specification of reactive behavior in the form of Event Condition Action (ECA) rules, allowing to give operational semantics for the enforcement of pre-obligations. Sorts and propositions of $\mathcal{L}_{\text{active}}$ are given in Tables I and II.

A state is a set of fluents. A fluent literal is either a fluent symbol or a fluent symbol preceded by $\neg$ ($\neg f$ is equivalent to $f$). The semantics of $\mathcal{L}_{\text{active}}$ defines a transition function which given a state and a (possibly empty) sequence of actions produces a new state as follows. Actions in the input sequence are processed successively. For every action, effect laws are evaluated and the fluent state is updated. If after the execution of the action, conditions in some event definition are true, the event is generated. The newly generated events trigger active rules. Identifiers of these triggered rules are added to the triggered rules set. When the last action in the input sequence is evaluated, if the triggered rules set is not empty, an action selection function selects the sequence of
A set of effect laws is consistent if it does not contain two effect laws for the same action which have contradictory effects and whose conditions are not disjoint. These conditions are verified by considering the ground instances of effect laws in the considered domain: If there is two effect laws “a causes f if \( p_1, \ldots, p_n \)” and “a causes g if \( q_1, \ldots, q_j, \ldots, q_m \),” then they should have either non-contradictory effects (\( f \neq -g \)) or disjoint conditions (\( \exists i, j : p_i = -q_j \)).

V. CONTEXT LANGUAGE AND CONTEXT MANAGEMENT

In the language, we separate the definition of security rule conditions from the definition of security rules using contexts [12]. A context defines a set of security rule conditions. The association of security rules with contexts allows the abstraction of complex conditions in the rule and thus, simplifies the interpretation of the policy. Contexts also allow context reuse in different security rules. In the following, we first show the definition of contexts then we present the management of the context state.

A. Context Rules

Security rule conditions define when some subject \( S \) is allowed, prohibited or obliged to take some action \( A \) on some object \( O \). Therefore, contexts enable the definition of constraints on the security rule triple \((S,A,O)\).

1) Context Rules: We consider context rules which are expressions of the following form:

\[
\text{Hold}_c(S,A,O,\text{start}(Ctx))
\]

after \( D_o(S',A',O') \)

if \( P_1, \ldots, P_n \)

Context rules define the moments at which the conditions identified by the context \( Ctx \) start and seize to be true for the subject \( S \), action \( A \) and object \( O \). More precisely, context rules for \( \text{start}(Ctx) \) define the conditions at which \( Ctx \) begins to hold. On the other hand, context rules for \( \text{end}(Ctx) \) specify when \( Ctx \) seizes to hold.

For instance, consider the following context rules.

\[
\text{Hold}_c(S,A,O,\text{start}(\text{in}_{\text{secure\_area}}))
\]

after \( D_o(S,\text{enter},L) \)

if \( \text{Classification}(L,\text{secure}) \)

\[
\text{Hold}_c(S,A,O,\text{end}(\text{in}_{\text{secure\_area}}))
\]

after \( D_o(S,\text{exit},L) \)

if \( \text{Classification}(L,\text{secure}) \)

These two rules specify that the persistent context \( \text{in}_{\text{secure\_area}} \) remains true for some subject \( S \) from the moment this subject enters a location which is classified \( \text{secure} \) until the moment the subject exists such location. These two moments are defined in terms of the event contexts \( \text{start}(\text{in}_{\text{secure\_area}}) \) and \( \text{end}(\text{in}_{\text{secure\_area}}) \).

2) State Context Rules: It is often convenient, particularly for authorization policies, to specify persistent security rule contexts in the form of state conditions. In this case, a security rule context is specified using propositions having the following form.

\[
\text{Hold}(S,A,O,Ctx) \leftarrow L_1, \ldots, L_n
\]

Where \( L_1, \ldots, L_n \) are conditions on the state. To support this form of context rules called state context rules, we transform state context rules into event context rules (given a domain description) [14]. In other words, we transform every state context rule into event context rules of the form \( \text{start}(Ctx) \) and \( \text{end}(Ctx) \). For instance, consider the following rule.

\[
\text{Hold}(S,A,O,\text{authenticated}_by_{\text{card}}) \leftarrow \text{Authenticated}(S,\text{card})
\]

The rule above specifies that the context \( \text{authenticated}_by_{\text{card}} \) holds for the subject \( S \) and any action/object while the fluent \( \text{Authenticated}(S,\text{card}) \) is true. Given this state context rule and the effect laws presented in Section IV-B, we use an algorithm [14] which transforms this state context into two event contexts \( \text{start}(Ctx) \) and \( \text{end}(Ctx) \). For instance, the following event context rules are derived for the context \( \text{authenticated}_by_{\text{card}} \).

\[
\text{Hold}_e(S,A,O,\text{start}(\text{authenticated}_by_{\text{card}}))
\]

after \( D_o(S,\text{authenticate}_by_{\text{card}},O) \)

if \( \text{Type}(O,\text{authentication_server}) \)

\[
\text{Hold}_e(S,A,O,\text{end}(\text{authenticated}_by_{\text{card}}))
\]

after \( D_o(S,\text{logs-off},\text{system}) \)

3) Temporal Contexts: Security rules are often associated with temporal conditions. These conditions are supported using temporal contexts. A temporal context is specified using the action \( \text{Clock} \). This action updates fluents which represent calendars available in the system, such as Minutes, Hours, Day, etc. Temporal contexts enable the specification of absolute and periodic temporal conditions. For instance, we may specify a temporal context \( \text{working-hours} \) which holds everyday from 8 until 18 as follows.

\[
\text{Hold}_e(S,A,O,\text{start}(\text{working-hours}))
\]

after \( \text{Clock} \)

if \( \text{Hours}(08) \)

\[
\text{Hold}_e(S,A,O,\text{end}(\text{working-hours}))
\]

after \( \text{Clock} \)

if \( \text{Hours}(18) \)

The specification of relative temporal deadlines in obligation policies are specified using the state context \( \text{delay}(\text{Nb.TimeUnit}). \) This state context holds for some security rule after the elapse of \( \text{Nb} \) time units after its activation. We also consider the composition of contexts [12] using the logic operators of conjunction (\&), disjunction (\( \oplus \)) and negation (\( \neg \)). The semantics of these operators is defined by the following rules.
\[ \text{Hold}(S, A, O, C_1 \land C_2) \leftarrow \\text{Hold}(S, A, O, C_1) \land \text{Hold}(S, A, O, C_2) \]
\[ \text{Hold}(S, A, O, C_1 \lor C_2) \leftarrow \text{Hold}(S, A, O, C_1) \lor \text{Hold}(S, A, O, C_2) \]
\[ \text{Hold}(S, A, O, C) \leftarrow \sim \text{Hold}(S, A, O, C) \]

**B. Management of the Context State**

In this paper, we consider persistent contexts. A persistent context \( Ctx \) holds from the moment the event \( \text{start}(Ctx) \) until the occurrence of \( \text{end}(Ctx) \). To enable the reasoning about which persistent contexts hold in every state, we consider that every persistent context \( Ctx \) is associated with a fluent \( \text{Hold}(S, A, O, Ctx) \). This fluent holds from the detection of the event context \( \text{start}(Ctx) \) until the occurrence of the event context \( \text{end}(Ctx) \). This is enforced using the following two active rules.

\[ \text{activate}_\text{Context} : \text{Hold}(S, A, O, \text{start}(Ctx)) \]
\[ \text{initiates} \text{Insert}(\text{Hold}(N, S, A, O, Ctx)) \]
\[ \text{deactivate}_\text{Context} : \text{Hold}(S, A, O, \text{end}(Ctx)) \]
\[ \text{initiates} \text{Remove}(\text{Hold}(N, S, A, O, Ctx)) \]

The rules above specify that the fluent \( \text{Hold}(S, A, O) \) should be inserted into (removed from, respectively) the state when \( \text{start}(Ctx) \) (\( \text{end}(Ctx) \) respectively) is detected. We consider the context state to be the subset of fluents which are of the form \( \text{Hold}(S, A, O, Ctx) \). In our framework, the context state is always updated before the evaluation of the policy. Therefore, the previous active rules are given higher priority than the rules which enforce the security policy presented later in Section VII.

**VI. SECURITY POLICY LANGUAGE**

We consider security rules which are close ground facts of the following form.

\[ \text{Permission}(N, SR, AA, OV, Ctx) \]
\[ \text{Obligation}(N, SR, AA, OV, Ctx, Ctx_v) \]

Where \( N \) is a rule identifier, \( SR \) is a subject or a role, \( AA \) is an action or an activity and \( OV \) is an object or a view. These expressions are called abstract security rules.

1) **Permission Rules**: A permission is associated with one state context \( (Ctx) \) called the permission context. A permission is effective only while this context is true, i.e. after the event context \( \text{start}(Ctx) \) occurs and before the event context \( \text{end}(Ctx) \) occurs. For example, consider the permission “cabinet members authenticated using their electronic cards may contact the president”. This permission is specified as follows:

\[ \text{Permission}(p, \text{cabinet members, contact, president, authenticated_by_card}) \]

This permission specifies that subjects assigned to the role of cabinet members may contact the president when the context \( \text{authenticated_by_card} \) is true.

2) **Obligation Rules**: Obligations are associated with two contexts: an obligation context \( (Ctx) \) and a violation context \( (Ctx_v) \). The obligation is effective while the context \( Ctx \) holds. It is violated if the context \( Ctx_v \) is detected while the obligation is effective. An obligation is enforced to be effective when it is fulfilled, *i.e.* when the subject executes the obliged action on the corresponding object.

For instance, consider the obligation “When the secretary of defense is in an insecure area, s/he should switch active radio communication devices to secure channels within 5 minutes”. This rule is specified as follows:

\[ \text{Obligation}(o_1, \text{secretary_of_defense, switch_sec_ch, comm_devices, active_device & ~in_secure_area, delay(5.minutes)}) \]

The context \( \text{active_devices} \) may be defined as follows.

\[ \text{Hold}(S, A, D, \text{start}(\text{active_device})) \]
\[ \text{after Do}(S, \text{activate_device, D}) \]
\[ \text{Hold}(S, A, D, \text{end}(\text{active_device})) \]
\[ \text{after Do}(S, \text{turnoff_device, D}) \]

**A. Specification of Pre-obligations**

A permission rule is contextual. For instance, permission \( p_1 \) in the previous Section VI specifies that cabinet are allowed to contact the president when they are in secure area and are authenticated by card. This contextual permission is enforced as follows in traditional systems: When a request to contact the president is made, if the cabinet is authenticated by card and in a secure area, the access is authorized. Otherwise, the request is denied. This enforcement model for authorizations may be sometimes too inflexible since it may be required to ask cabinet members to authenticate after their access request (if they are not authenticated).

To simplify the specification that some requirements should be evaluated dynamically, we associate every user-defined context in the policy \( Ctx \) with another context denoted \( d_{Ctx} \), called the dynamic version of \( Ctx \). When some context \( d_{Ctx} \) is used in some security rule, it is interpreted as a requirement that may be fulfilled dynamically after the access request. For instance, consider our example. To specify that cabinet members should be asked to authenticate after they request to contact the president if they are not authenticated, we specify a permission rule using the context \( d_{\text{authenticated_by_card}} \) as follows:

\[ \text{Permission}(p_1, \text{cabinet_members, contact, president, d_{\text{authenticated_by_card}}}) \]

In this case, when some cabinet member contacts the president and s/he is not authenticated. The cabinet member is asked to authenticate by card. This requirement is enforced using an obligation for the cabinet member to authenticate by card. When this obligation is fulfilled, access is allowed.

It is necessary to associate every obligation with a deadline condition. For instance, it may be required to specify that the cabinet member should authenticate within
In our work, we allow the association of every dynamic context with a deadline in the form of an attribute Violation. For instance, we specify that the obligation associated with the context $d_{\text{authenticated by card}}$ has a deadline of 5 minutes by updating the value of its attribute Violation to \text{Violation}($d_{\text{authenticated by card}}, \text{delay}$(5\text{ minutes}))

For every dynamic context, a default deadline defined by the policy administrator is used unless this attribute is updated. In Section II, we have also argued that it should be possible to enable the association of pre-obligations with weights to enable the selection of the simplest set of pre-obligations for a given access request. For this reason, we consider a second attribute Weight for dynamic contexts. The default value of this attribute is 1.

For instance, consider the following permission.

\[
\text{Permission}(p_2, \text{secretary of defense, contact, president} \rightarrow d_{\text{authenticated by password}} \land d_{\text{in secure area}})
\]

Where the context $\text{authenticated by password}$ is defined similarly to the context $\text{authenticated card}$ and the context $\text{in secure area}$ is defined as in Section V. Assume that it is considered that authentication using password is simpler than authentication by card. In this case, we may assign the contexts $d_{\text{authenticated by password}}$ and $d_{\text{authenticated by card}}$ the weights of 2 and 3 respectively. Consider also that the context $d_{\text{in secure area}}$ is given a more important weight of 4. In this setting, when the secretary of defense requests to contact the president, given a more important weight of 4. In this setting, when the secretary of defense requests to contact the president, there are several possibilities. For instance, if $s \neq \text{he}$ is not authenticated but is in a secure location, $s \neq \text{he}$ is asked to authenticate by password. If $s \neq \text{he}$ is not authenticated nor is in a secure location, $s \neq \text{he}$ is asked to authenticate by card.

VII. POLICY MANAGEMENT & ENFORCEMENT

We distinguish between abstract and concrete security rules as follows: Abstract policy rules describe the global system policy and is specified by policy administrators. Concrete rules, on the other hand, are the security rules which are derived from the abstract policy as follows.

\[
\text{Permission}(N, S, A, O, Ctx) \leftarrow \text{Permission}(N, R, AA, V, Ctx), \text{Empower}'(S, R), \text{Consider}'(A, AA), \text{Usse}'(O, OV)
\]

The predicate $\text{Empower}'(S, SR)$ specifies that $S$ should be either $SR$ if $SR$ is a subject or a subject empowered into the role of $SR$ if $SR$ is a role. It is specified as follows.

\[
\text{Empower}'(S, SR) \leftarrow \text{Subject}(SR)
\]

\[
\text{Empower}'(S, SR) \leftarrow \text{Role}(SR), \text{Empower}(S, SR)
\]

Similarly, the predicate $\text{Consider}'(A, AA)$ states that $A$ should be either the action $AA$ if $AA$ is an action or an action considered in $AA$ if $AA$ is an activity. The predicate $\text{Use}'(O, OV)$ dictates that $O$ should be either the object $OV$ or an object used in $OV$ if $OV$ is a view. We also derive concrete obligation rules for individual subjects, actions and objects from abstract obligation rules. In the following, we formalize and provide operational semantics for the policy management of concrete obligation rules using active rules.

A. Permission Activation and Deactivation

Every concrete permission rule is associated with a context which defines when it is effective. We therefore associate every permission in the state with a fluent $\text{Permitted}(N, SA, O, Ctx)$. This fluent starts to hold when the permission’s context begins to hold. It seizes to hold when the permission’s context is ended. This is specified using the following active rules:

\[
\text{activate}_{\text{Permission}} : \text{Hold}_e(S, A, O, \text{start}(Ctx))
\]
\[
\text{initiates}_{\text{Insert}}(\text{Permitted}(N, S, A, O, Ctx))
\]
\[
\text{if } \text{Permission}(N, S, A, O, Ctx)
\]
\[
\text{deactivate}_{\text{Permission}} : \text{Hold}_e(S, A, O, \text{end}(Ctx))
\]
\[
\text{initiates}_{\text{Remove}}(\text{Permitted}(N, S, A, O, Ctx))
\]
\[
\text{if } \text{Permitted}(N, S, A, O, Ctx)
\]

The rules above specify that the action $\text{Insert}(\text{Permitted}(N, SA, O, Ctx))$ should be taken when the context of some permission’s is activated. This action makes the fluent $\text{Permitted}$ hold as specified in the following effect law.

\[
\text{Insert}(\text{Permitted}(N, S, A, O, Ctx))
\]
\[
\text{causes } \text{Permitted}(N, S, A, O, Ctx)
\]

Reciprocally, we specify that $\text{Permitted}$ seizes to hold after the execution of $\text{Remove}$ for the fluent $\text{Permitted}$. In the policy, an access may be authorized by more than one permission. Therefore, we consider an additional fluent $\text{Permitted}(S, A, O)$ which holds for some access $(S, A, O)$ while this access is allowed. This fluent begins to hold for $(S, A, O)$ whenever some permission for $(S, A, O)$ is activated. It seizes to hold after the deactivation of a permission for $(S, A, O)$ only if there is no other permission for $(S, A, O)$ in the state.

B. Obligation Activation and Deactivation

To manage obligations, we associate every concrete obligation with a fluent $\text{Obliged}(N, SA, O, Ctx, Ctx_v)$. This fluent represents that there is an effective obligation for $S$ to take $A$ on $O$ before $Ctx_v$. An obligation is said to be deactivated when its context $Ctx$ is ended while it is effective. When an obligation is deactivated, the fluent $\text{Obliged}$ seizes to hold. This is formalized using the following two active rules.

\[
\text{activate}_{\text{Obligation}} : \text{Hold}_e(S, A, O, \text{start}(Ctx))
\]
\[
\text{initiates}_{\text{Insert}}(\text{Obligated}(N, S, A, O, Ctx, Ctx_v))
\]
\[
\text{if } \text{Obligation}(N, S, A, O, Ctx, Ctx_v)
\]
\[
\text{deactivate}_{\text{Obligation}} : \text{Hold}_e(S, A, O, \text{end}(Ctx))
\]
\[
\text{initiates}_{\text{Remove}}(\text{Obligated}(N, S, A, O, Ctx, Ctx_v))
\]
\[
\text{if } \text{Obligation}(N, S, A, O, Ctx, Ctx_v)
C. Obligation Fulfillment and Violation

As opposed to permissions, obligations may additionally be violated and fulfilled. An effective obligation is fulfilled when its required action is taken. Actions required by effective obligations are monitored using the following context.

\[ Hold_d(S, A, O, \text{start}(ctx_{fulfillment})) \]
\[ \text{after} \; Do(S, A, O) \]
\[ \text{if} \; \text{Obligated}(N, S, A, O, Ctx, Ctx_v) \]

The context \( ctx_{fulfillment} \) holds for some \((S,A,O)\) when the action \( Do(S,A,O) \) is taken and there is an effective obligation requiring \((S,A,O)\). When \( \text{start}(S,A,O,\text{start}(ctx_{fulfillment})) \) is detected, effective obligations for \((S,A,O)\) are fulfilled.

\[ \text{fulfill\_Obligation} : \; Hold_d(S, A, O, \text{start}(ctx_{fulfillment})) \]
\[ \text{initiates} \; \text{Fulfill}(N, S, A, O) \]
\[ \text{if} \; \text{Obligated}(N, S, A, O, Ctx, Ctx_v) \]

Reciprocally, the detection of the deadline context of an effective obligation violates this obligation. This is specified as follows.

\[ \text{violate\_Obligation} : \; Hold_d(S, A, O, \text{start}(ctx_v)) \]
\[ \text{initiates} \; \text{Violate}(N, S, A, O) \]
\[ \text{if} \; \text{Obligated}(N, S, A, O, Ctx, Ctx_v) \]

The actions \( \text{Fulfill} \) and \( \text{Violate} \) indicate the fulfillment and violation of obligations respectively. In this paper, we assume for simplicity that obligations are deactivated whenever they are violated/fulfilled. Therefore, the fluent \( \text{Obligated} \) seizes to hold when the actions \( \text{Fulfill} \) and \( \text{Violate} \) are taken.

VIII. DERIVATION OF DYNAMIC CONTEXTS

In our work, we consider that every user-specified context \( Ctx \) has corresponding dynamic context \( d_Ctx \). This simplifies the specification of the policy by enabling policy administrators to easily choose whether any context should be statically or dynamically evaluated.

Dynamic contexts and their associated pre-obligations are automatically derived from the definition of user-specified contexts using algorithm 1. Algorithm 1 takes the set of user-defined event context definitions \( C \) as input. It produces, if possible, the definition of the dynamic contexts \( d_C \). It also derives for every dynamic context an obligation rule \( O(d_C) \). This obligation \( O(d_C) \) defines the action which should be taken for the context \( d_C \) to be activated. The fulfillment of this obligation activates the context \( d_C \) (as well as the context \( C \)) for the access requester.

The algorithm verifies every user-defined event context rule as follows. First, if the context is of the form \( \text{start}(C) \) and is started by some action \( A \) (line 5), then a dynamic context \( d_C \) is defined similarly to \( C \), i.e. \( d_C \) is associated with the same actions which start and end \( C \) (lines 7-10).

An obligation is then constructed. The obligation’s identifier is \( O(d_C) \) (line 12). Its role and view are initialized using the role \( \text{any\_subject} \) and the view \( \text{any\_object} \) (line 13). These entities represent all subjects and all objects in the system respectively.

Constraints over the parameters of the action which starts \( C \) in the \( \text{after} \) part (lines 13-14) and in the \( \text{if} \) part (lines 15-19) of the context definition of \( \text{start}(C) \) are then checked. If some constraint over the action’s subject or object \((S,A,O)\) is specified, it is used as the subject/role and object/view of the obligation respectively.

An event context identifier of the form \( \text{start}(O(d_C)) \) is then used to denote the activation conditions of the obligation (line 20). The context \( \text{start}(O(d_C)) \) is then defined (lines 21-23). Its definition states that it should be detected after the execution of the action \( \text{Find\_Obligations}(S',A',O') \), if the fluent \( \text{Pre\_Obl\_For\_Access}(O(d_C),S,A,O,S',A',O') \) holds. The action \( \text{Find\_Obligations} \) checks the policy for possible pre-obligations when the access \((S',A',O')\) is not authorized. The fluent \( \text{Pre\_Obl\_For\_Access} \), on the other hand, denotes that the obligation associated with \( d_C \) for the subject \( S \) to take the action \( A \) on the object \( O \) has been selected for the authorization of \((S',A',O')\). The selection of pre-obligations is detailed in Section IX.

Finally, if a user-specified \( \text{Violation\_Context} \) attribute for \( d_C \) exists, it is used as the obligation violation context. Otherwise, the default context is used (lines 24-27).

The algorithm returns the constructed event definitions, obligation rules and context attributes. These elements are added to the policy.

For instance, the application of the algorithm to the context \( \text{start}(\text{authorized\_by\_card}) \) presented in Section V produces: (1) the dynamic context definition for \( d_{\text{authorized\_by\_card}} \) defined similarly to the user-defined context \( \text{authorized\_by\_card} \) (2) The context attribute \( \text{Type}(d_{\text{authorized\_by\_card}},\text{dynamic}) \) and (3) The obligation and the event context rule specified below.

\[ \text{Obligation}(O(d_{\text{authorized\_by\_card}}), \text{any\_subject}, \text{authorized\_by\_card}, \text{any\_object}, \text{start}(\text{O(d_{authorized\_by\_card}}), \text{delay}(3, \text{Minutes})) \]
\[ \text{Hold}_d(S, \text{authorized\_by\_card}, O, \text{start}(O(d_{\text{authorized\_by\_card}}))) \]
\[ \text{after} \; \text{Find\_Obligations}(S', A', O') \]
\[ \text{if} \; \text{Pre\_Obl\_For\_Access}(O(d_{\text{authorized\_by\_card}}), S, \text{authorized\_by\_card}, O, S', A', O')) \]

The obligation rule defines an obligation \( O(d_{\text{authorized\_by\_card}}) \) which states that the action \( \text{authorized\_by\_card} \) should be taken by any subject on any object when the context \( \text{start}(d_{\text{authorized\_by\_card}}) \) is detected. This context is detected for the subject \( S \) and object \( O \) if \( S \) and \( O \) were selected to fulfill the obligation after the execution of the action \( \text{Find\_Obligations} \) for the access request \((S',A',O')\), as indicated by the fluent \( \text{Pre\_Obl\_For\_Access}(O(d_{\text{authorized\_by\_card}}), S, \text{authorized\_by\_card}, O, S', A', O')) \).
### Algorithm 1: Dynamic Context Derivation

<table>
<thead>
<tr>
<th>Input</th>
<th>(1) Event Definitions ( E )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(2) A Set of Event Context Definitions (ED)</td>
</tr>
<tr>
<td></td>
<td>(3) A Set of Obligation Rules (Obl)</td>
</tr>
<tr>
<td></td>
<td>A Set of context attributes (CA)</td>
</tr>
<tr>
<td>begin</td>
<td></td>
</tr>
<tr>
<td>ED = Obl = CA = ( \emptyset );</td>
<td></td>
</tr>
<tr>
<td>foreach User-defined Event Context Definition of the form</td>
<td></td>
</tr>
<tr>
<td>“Hold(_{(S',A',O',C)}) after Do(S,A,O) if ( P_1, ..., P_n )” in ( E ) do</td>
<td></td>
</tr>
<tr>
<td>if ( Ctx = \text{start}(C) ) and ( A ) is an action then</td>
<td></td>
</tr>
<tr>
<td>( ED = ED \cup { \text{Hold}(S',A',O',\text{start}(\text{d}_C)) \text{ after } Do(S,A,O) \text{ if } P_1, ..., P_n } )</td>
<td></td>
</tr>
<tr>
<td>foreach User-defined Context in ( E ) of the form</td>
<td></td>
</tr>
<tr>
<td>“Hold(_{(S',A',O',end(\text{d}_C))}) after Do(S,A,O) if ( Q_1, ..., Q_m )” do</td>
<td></td>
</tr>
<tr>
<td>( ED = ED \cup { \text{Hold}(S',A',O',\text{end}(\text{d}_C)) \text{ after } Do(S,A,O) \text{ if } Q_1, ..., Q_m } )</td>
<td></td>
</tr>
<tr>
<td>( N' = \text{O}(d_C) )</td>
<td></td>
</tr>
<tr>
<td>( SR' = \text{any}_subject; OV' = \text{any}_object )</td>
<td></td>
</tr>
<tr>
<td>if ( S ) is a subject then ( SR' = S;OV' = S )</td>
<td></td>
</tr>
<tr>
<td>if ( O ) is an object then ( OV' = O;SR' = O )</td>
<td></td>
</tr>
<tr>
<td>foreach Condition ( P_i ) in ( P_1, ..., P_n ) do</td>
<td></td>
</tr>
<tr>
<td>if ( P_i \equiv (S = SR) ) or ( P_i \equiv \text{Empower}(S,SR) ) then</td>
<td></td>
</tr>
<tr>
<td>( SR' = SR;OV' = OV )</td>
<td></td>
</tr>
<tr>
<td>else if ( P_i \equiv (O = OV) ) or ( P_i \equiv \text{Use}(O,OV) ) then</td>
<td></td>
</tr>
<tr>
<td>( SR' = SR;OV' = OV )</td>
<td></td>
</tr>
<tr>
<td>( Ctx' = \text{start}(\text{d}_C) );</td>
<td></td>
</tr>
<tr>
<td>( ED = ED \cup { \text{Hold}(S,A,O,\text{start}(\text{d}_C)) } )</td>
<td></td>
</tr>
<tr>
<td>\text{After Find Obligations}(S',A',O')</td>
<td></td>
</tr>
<tr>
<td>if ( \text{PreOblForAccess}(\text{O}(\text{d}_C),S,A,O,S',A',O') } )</td>
<td></td>
</tr>
<tr>
<td>( Ctx_i' = Ctx_i )</td>
<td></td>
</tr>
<tr>
<td>else ( Ctx_i' = \text{DefaultVIolation}_Context )</td>
<td></td>
</tr>
<tr>
<td>Obl = Obl \cup { Obligation(N',SR',AO',OV',Ctx_i',Ctx_i') }</td>
<td></td>
</tr>
<tr>
<td>return ( (CA,ED,obl) )</td>
<td></td>
</tr>
</tbody>
</table>

### IX. Enforcement of Pre-obligation Policies

In this section, we formalize policy enforcement shown in Figure 1: When an access request is made, access is granted if it is authorized by an effective permission. Otherwise, the authorization policy is checked for pre-obligations which would allow the access. If none is found, access is denied. If pre-obligations are activated, they are enforced as follows. Whenever an effective permission for the requested access is activated or if one of the pre-obligations is violated/deactivated, pre-obligations are deactivated. When all pre-obligations are successfully fulfilled, access is granted.

#### A. Authorization Policy Enforcement

First, we specify the context access\(_{req}\_ctx\) as follows.

\[
\text{Hold}_{(S,A,O,\text{start}(access\_req\_ctx))}
\]

\text{after Request}(S,A,O)

The context access\(_{req}\_ctx\) holds for \((S,A,O)\) after the occurrence of the special action Request\((S,A,O)\). This action represents that \(S\) has requested to take \(A\) on \(O\). It holds until this access request is honored, \textit{i.e.} when the access is either allowed or denied. The end of access\(_{req}\_ctx\) is therefore specified as follows.

\[
\text{Hold}_{(S,A,O,\text{end}(access\_req\_ctx))}
\]

\text{after Allow}(S,A,O)

\[
\text{Hold}_{(S,A,O,\text{end}(access\_req\_ctx))}
\]

\text{after Deny}(S,A,O)

When an access request is made, it is directly granted if it is authorized by an effective permission. This is specified using the following rule.

\[
\text{allow\_Access} : \text{Hold}_{(S,A,O,\text{start}(access\_req\_ctx))}
\]

\text{initiates} \text{Allow}(S,A,O)

If there is no effective permission for the requested access, we check the authorization policy for pre-obligations.

\[
\text{find\_Obligations} : \text{Hold}_{(S,A,O,\text{start}(access\_req\_ctx))}
\]

\text{initiates} \text{Find\_Obligations}(S,A,O)

If \(\neg\text{Permitted}(S,A,O)\) the action Find\_Obligations selects (if possible) the simplest set of pre-obligations required to allow \((S,A,O)\) by executing the algorithm 2. This algorithm works as follows: it checks every permission which permits \((S,A,O)\). First, the permission context \(Ctx\) is transformed into the disjunctive normal form (DNF) to identify the expressions \((CN_i)\) of basic contexts which allow the requested access when these basic contexts hold simultaneously. More precisely, we consider that Ctxt are transformed into a sentence of the form of \((\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} C_{ij})\) where every \(C_{ij}\) is a basic non-composed context. Any conjunction of contexts \((CN_i = \bigwedge_{j=1}^{m_i} C_{ij})\) in this sentence is considered valid if: (1) all its non-dynamic contexts are true, (2) each of its dynamic contexts which does not hold can be activated. A dynamic context \(C\) can be activated if there exists \((S',A',O')\) and a dynamic event definition \(\text{Hold}_{(S,A,O,\text{start}(C))}\) such that the conditions of this event definition are true. This ensures that when \(Do(S',A',O')\) is preformed, \(C\) is activated. For every inactive dynamic context which can be activated, a fluent of the form Pre\_Obl\_For\_Access(\(\text{O}(C),S',A',O',S,A,O)\) is added to the set Obligations. This fluent specifies that \(Do(S',A',O')\) should be taken to activate the dynamic context \(O(C)\) and, subsequently allow the requested access \((S,A,O)\). Then, the weight assigned with \(O(C)\) is added to the \(CN\_Weight\). After the evaluation of every \(CN_i\), if the sum of the weights of its pre-obligations \(CN\_Weight\) is less than the minimum weight Min\_Weight, the pre-obligations of this \(CN_i\) are selected. After the evaluation of the authorization policy, the algorithm returns No\_Pre\_Obligations\_For\((S,A,O)\) if no pre-obligations are possible for the access. Otherwise, the set of pre-obligations selected for the access is returned.

If no pre-obligations are returned after the execution of Find\_Obligations, the context no\_pre\_obligations holds and
access is denied. We specify the denial of access as follows.

\[ \text{Hold}_d(S, A, O, \text{start}(\text{no pre-obligations}) \]
\[ \text{after Find Obligations}(S, A, O) \]
\[ \text{if } \text{No Pre Obl For}(S, A, O) \]
\[ \text{Hold}_d(S, A, O, \text{start}(\text{no pre-obligations})) \]
\[ \text{initiates Deny}(S, A, O) \]

The fluent \text{No Pre Obl For}(S,A,O) as well as the context \text{no pre-obligations} seize to hold when the access request is honored to allow the revaluation of the authorization policy at subsequent access requests.

**Algorithm 2: Pre-obligation Selection**

\[
\begin{align*}
\text{Input} & : (1) \text{access Request}(S,A,O) \\
& \quad (2) \text{The Authorization Policy } \mathcal{P} \\
& \quad (3) \text{Event Definitions } E \\
\text{Output} & : \text{Selected Pre-obligations} \\
\text{begin} & \text{Selected} = \emptyset; \text{Min}\_\text{Weight} = \infty; \\
& \text{foreach } \text{Permission}(S,A,O,Ctx) \in \mathcal{P} \text{ do} \\
& \quad \text{Ctx} = \text{DNF}(\text{Ctx}); \\
& \quad \text{foreach } \text{Conjunction } \text{CN} \text{ of } \text{the form } \bigwedge_{j=1}^{n} \text{C}_j \text{ in } \text{Ctx} \text{ do} \\
& \quad \quad \text{CN}_\text{Weight} = 0; \quad \text{Obligations} = \emptyset; \quad \text{Valid} = \text{true}; \\
& \quad \quad \text{foreach Basic Context } \text{C}_j \text{ in } \text{CN} \text{ do} \\
& \quad \quad \quad \text{if } \sim \text{Type}(\text{C}_j, \text{dynamic}) \& \sim \text{Hold}(S,A,O,C_j) \text{ then} \\
& \quad \quad \quad \quad \text{Valid} = \text{false}; \\
& \quad \quad \quad \text{if } \text{Type}(\text{C}_j, \text{dynamic}) \& \sim \text{Hold}(S,A,O,C_j) \text{ then} \\
& \quad \quad \quad \quad \text{if } (S^{3}\sim S^{3} \sim S^{0}) \& \sim \text{Hold}(S,A,O,start(C_j)) \text{ after} \\
& \quad \quad \quad \quad \text{Do}(S' ,A',O') \text{ if } p_1,\ldots,p_n \text{ hold} \text{ then} \\
& \quad \quad \quad \quad \quad \text{Obligations} = \text{Obligations} \cup \text{Pre Obl For Access}(O(C_j),S',A',O',S,A,O); \\
& \quad \quad \quad \quad \text{if } \text{Weight}(C_j, X) \text{ then} \\
& \quad \quad \quad \quad \quad \text{CN}_\text{Weight} = \text{CN}_\text{Weight} + X; \\
& \quad \quad \quad \text{else} \\
& \quad \quad \quad \quad \quad \text{Valid} = \text{false}; \\
& \quad \quad \quad \text{if } \text{Valid} = \text{true} \& \text{Min}_\text{Weight} > \text{CN}_\text{Weight} \text{ then} \\
& \quad \quad \quad \quad \text{Selected} = \text{Obligations}; \\
& \quad \quad \quad \quad \quad \text{Min}_\text{Weight} = \text{CN}_\text{Weight}; \\
& \quad \text{if } \text{Selected} = \emptyset \text{ then} \\
& \quad \quad \text{Selected} = \{ \text{No Pre Obligations For}(S,A,O) \} \\
\text{return Selected}; \\
\text{end}
\end{align*}
\]

**B. Enforcement of Pre-obligation Sets**

After the activation of a set of pre-obligations for an access \((S,A,O)\), pre-obligations are enforced as follows.

**Permission Activation:** Whenever a permission is activated for \((S,A,O)\) and there is a request to take \((S,A,O)\), pre-obligations for \((S,A,O)\) are deactivated and access is allowed. The following context starts to hold when some requested access \((S,A,O)\) become authorized.

\[ \text{Hold}_d(S, A, O, \text{start}(\text{authorized request}) \]
\[ \text{after Insert}(\text{Permitted}(N, S, A, O)) \]
\[ \text{if Permitted}(S, A, O, \text{access_request_ctx}) \]

When an access request for \((S,A,O)\) is authorized, the following rule deactivate pending pre-obligations for \((S,A,O)\) (if any exists).

\[ \text{deactivate Pre}: \text{Hold}_d(S, A, O, \text{start}(\text{authorized request}) \]
\[ \text{initiates Remove(Pre Obl For Access}(N, S', A', O', S, A, O)) \]
\[ \text{if Pre Obl For Access}(N, S', A', O', S, A, O) \]

We also accept the requested access by initiating the action \text{Allow} using the following active rule.

\[ \text{allow access}: \text{Hold}_d(S, A, O, \text{start}(\text{authorized request})) \]
\[ \text{initiates Allow}(S, A, O) \]

**Violation of Pre-obligations:** When a pre-obligation is violated, the fulfillment of other pre-obligations becomes unnecessary since the access will not be allowed. Therefore, we deactivate in this case other related pre-obligations \((i.e., \text{pre-obligations for the same access request})\) and deny the access. We define the context \text{pre obl violated} which holds when pre-obligations are violated as follows.

\[ \text{Hold}_d(S, A, O, \text{start}(\text{pre obl violated}) \]
\[ \text{after Violate}(N, S', A', O') \]
\[ \text{if Pre Obl For Access}(N, S', A', O', S, A, O) \]

When \text{pre obl violated} starts to hold, the following two rules deactivate pre-obligations and deny the access requested.

\[ \text{violate pre}: \text{Hold}_d(S, A, O, \text{start}(\text{pre obl violated}) \]
\[ \text{initiates Remove(Pre Obl For Access}(N, S', A', O', S, A, O)) \]
\[ \text{if Pre Obl For Access}(N, S', A', O', S, A, O) \]

\[ \text{deny access}: \text{Hold}_d(S, A, O, \text{start}(\text{pre obl violated}) \]
\[ \text{initiates Deny}(S, A, O) \]

**Pre-obligation Fulfillment:** When pre-obligations are fulfilled, they are removed from the state using the following active rule.

\[ \text{fulfill pre}: \text{Hold}_d(S, A, O, \text{start}(\text{ctx fulfillment}) \]
\[ \text{initiates Remove(Pre Obl For Access}(N, S, A, O, S', A', O')) \]
\[ \text{if Pre Obl For Access}(N, S, A, O, S', A', O') \]

**X. Application Example**

To illustrate the concepts presented in this paper and discuss the evolution of the authorization and obligation policy states when pre-obligations are supported, we consider the example policy specified in Section VI-A. This policy includes the following permission rules.

\[ \text{Permission}(p_1,\text{cabinet_members,contact,\text{president,}} \]
\[ \quad \text{d\_authenticated\_by\_card}) \]
\[ \text{Permission}(p_2,\text{secretary\_of\_defense,contact,\text{president,}} \]
\[ \quad \text{d\_in\_secure\_area \& d\_authenticated\_by\_password}) \]
Table III shows the values given to the attributes Weight and Violation of each context. We now first discuss the selection of pre-obligations after the secretary of defense requests to contact the president in the following situations:

- (S1) The secretary is authenticated by card: Access is directly granted since the permission \( p_1 \) is effective.
- (S2) The secretary of defense is authenticated by password and is not in a secure location: The secretary of defense is asked to authenticate by card since this pre-obligation is assigned lower weight than the weight given to the obligation to move to a secure location.
- (S3) The secretary of defense is not authenticated and is in a secure area: The secretary of defense is asked to authenticate by password.

Assume we replace the permission \( p_1 \) in the policy above with the following permission.

\[
\text{Permission}(p_3, \text{cabinet Members}, \text{contact,president, working hours & \& \& authenticated by card})
\]

The permission \( p_3 \) specifies that cabinet members may contact the president during working hours provided that they are authenticated by card. In this case, the selection of pre-obligations proceeds as follows:

- (S4) During working hours, the secretary of defense is not authenticated nor is in a secure area and requests to contact the president: The secretary of defense is asked to authenticate by card.
- (S5) Outside of working hours, the secretary of defense is not authenticated nor in a secure area and requests to contact the president: The secretary of defense is asked to authenticate by password and to move to a secure location since only \( p_3 \) can be activated.

Tables IV and V show the selection of pre-obligations and discusses the evolution of the state of authorizations and obligations in the different situations just described. Each table row represents the state obtained by the execution of the action appearing in the rightmost column of the row above. The obligations to authenticate by card, to authenticate by password and to move to a secure location are denoted \( o_c \), \( o_p \) and \( o_{sa} \), respectively. We only give identifiers for situations when it is necessary. We will now first consider the evolution of the authorization and obligation policy states for the situation (S3) where the secretary of defense is asked to authenticate by password (within 3 minutes). In this scenario, the following may occur.

- The secretary of defense authenticates by password successfully and access is granted.
- The secretary of defense authenticates by card. In this case, the permission \( p_1 \) is activated and the obligation to authenticate by password is deactivated.
- The secretary of defense fails to authenticate within 3 minutes and access is denied.
- After authentication, the secretary of defense logs off the system. When this happens, the secretary of defense is no longer authenticated and permission \( p_2 \) is deactivated.

We now consider the situation (S5) where the secretary of defense is asked to authenticate by password (within 3 minutes) and to move to a secure area (within 5 minutes). The following may happen.

- The secretary of defense successfully fulfills the two pre-obligations. In this case, the permission \( p_2 \) will be activated and access is granted.
- The secretary of defense fails to authenticate within 3 minutes. In this case, the second pre-obligation to move to a secure area is deactivated and access is denied.
- The secretary of defense authenticates within 3 minutes but fails to move to a secure area within 5 minutes. In this case, access is denied.

### XI. Related Work

Other models have been proposed to support pre-obligations in access control policies.

To our knowledge, the notion of provisional actions was first introduced by Kudo and Hada in [15] to enable the association of access control security rules for XML documents with actions that should be triggered by access requests. In [16], multiple hierarchies and property propagation are studied. In contrast, we study provisional actions in the form of user-actions which are monitored for fulfillment/violation and formalize policy enforcement and evolution.

In [17], the ASL access control language [3] is extended to allow the association of security rules with provisional actions. An architecture for the enforcement of these provisional actions is proposed. Bettini et al. [5], [6] study the association of access control rules with provisions and obligations and propose algorithms for the computation a minimal provisions and obligations set. Obligation definition and monitoring in the framework is discussed in [18]. In comparison, the main advantage of our work is that we consider a formal description of change in state using the concepts of action specification languages. This enables us to formalize the activation/deactivation/violation/fulfillment of pre-obligations and the effects of these operations on the authorization and obligation states. Consequently, we clarify the semantics of pre-obligations by giving their enforcement declarative semantics. In addition, our formal model for pre-obligations is given operational semantics using ECA rules.

In [7], an obligation model supporting the specification of pre- and post-obligations is presented. The paper studies two interactions between permissions and obligations namely invalid permission due to obligation cascading and the dominance of obligations. With respect to our work, the model subordinates obligations to permissions, only considers temporal obligation deadlines and does not consider the selection of pre-obligations after access requests. Furthermore, in our policy language, obligations are specified separately from access control rules and provisions.
are specified in the form of contexts in permission rules (as opposed to the specification of obligations embedded in access control rules). This simplifies the representation of the access control policy and, additionally, enables us to support general obligations which do not depend on access requests. Moreover, we formalize the enforcement of pre-obligations and provide declarative semantics for their enforcement.

The UCON model [19] introduces obligations to deal with usage control requirements and introduces the notion of attribute mutability. The model is formalized in [20]. Pre-obligations in UCON are evaluated using the functional predicate “preB” which checks, when an access request is made, whether pre-obligations required for this access have been fulfilled. Formally, checking pre-obligation fulfillment in UCON is similar to checking regular permission conditions. By contrast, in our framework, pre-obligations are activated just after an access request if their fulfillment is required to enable the access. This is an important advantage since, whenever necessary, subjects may be assisted by the system in accessing resources. Additionally, the UCON model does not support the specification of general or global obligations since obligations are always associated with resource usage.

Other works on trust management [8], [21], [22] studied the use of abduction in explaining access denials to users by searching for missing facts or credentials which would allow the requested access. The work that is most relevant to ours is [23] where a logic and an inference system for reasoning about sequences of user-actions and their effects on the authorization policy state are presented. This work
is complementary to ours since we essentially study the enforcement and management of pre-obligations as opposed to how to derive the actions that should be taken to obtain particular permissions.

XII. CONCLUSION

In this paper, we studied the specification, selection and enforcement of pre-obligations. First, we have proposed to specify pre-obligations in access control rules in the form of permission contexts to simplify both the specification and interpretation of the access control policy. We have also considered the notion of dynamic context attributes to allow the association of dynamic contexts (denoting pre-obligations) with different weights and deadlines. We have then studied the selection of pre-obligations after access requests and formalized the enforcement of pre-obligations and its effects on the policy state.

Future work consists of modeling consent requirements in the form of special pre-obligations and the integration of group pre-obligations [24] to enable the specification of pre-obligations which may be fulfilled in different ways.

REFERENCES


