Access control and obligations in the category-based metamodel: a rewrite-based semantics[⋆]

Sandra Alves[1], Anatoli Degtyarev[2], and Maribel Fernández[2]

[1] Dept. of Computer Science, University of Porto, Porto, Portugal
[2]King's College London, Dept. of Informatics, London WC2R 2LS, U.K.

Abstract. We define an extension of the category-based access control (CBAC) metamodel to accommodate a general notion of obligation. Since most of the well-known access control models are instances of the CBAC metamodel, we obtain a framework for the study of the interaction between authorisation and obligation, such that properties may be proven of the metamodel that apply to all instances of it. In particular, the extended CBAC metamodel allows security administrators to check whether a policy combining authorisations and obligations is consistent.
Key Words: Security Policies, Access Control, Obligations, Rewriting

1 Introduction

Access control policies specify which actions users are authorised to perform on protected resources. An authorisation may entail an obligation to perform another action on the same or another resource. Standard languages for the specification of access control policies include also a number of primitives to specify obligations associated with authorisations. For example, within XACML [19], an obligation is a directive from the Policy Decision Point (PDP) to the Policy Enforcement Point (PEP) specifying an action that must be carried out before or after an access is approved.

The notion of obligation helps bridge a gap between requirements and policy enforcement. For example, consider a hospital scenario in which any doctor may be authorised to read the medical record of a patient in an emergency situation, but in that case there is a requirement to inform the patient afterwards. Although access control models deal mainly with authorisations, incorporating obligations facilitates the analysis of compatibility between obligations and authorisations.

[⋆] This work was partially funded by the European Office of Aerospace Research and Development (EOARD-AFOSR).

A metamodel of access control, which can be specialised for domain-specific applications, has been proposed in [3]. It identifies a core set of principles of access control, abstracting away many of the complexities that are found in specific access control models, in order to simplify the tasks of policy writing and policy analysis. The metamodel focuses on the notion of a *category*, which is a class of entities that share some property. Classic types of groupings used in access control, like a role, a security clearance, a discrete measure of trust, etc., are particular instances of the more general notion of category. In category-based access control (CBAC), permissions are assigned to categories of users, rather than to individual users. Categories can be defined on the basis of user attributes, geographical constraints, resource attributes, etc. In this way, permissions can change in an autonomous way (e.g., when a user attribute changes), unlike, e.g., role-based models [1], which require the intervention of a security administrator.

In this paper, we define an extension of the CBAC metamodel to accommodate a general notion of an *obligation*, obtaining a formal framework for modelling and enforcing access control policies that involve authorisations and obligations. We show examples of application in the context of emergency management. We do not make any specific assumptions on the components of the system. Instead, we aim at defining an abstract model of access control and obligations that can be instantiated in various ways to satisfy specific requirements. To specify dynamic policies involving authorisations and obligations in the metamodel, we adjust the notion of *event* given in [15] and describe a set of core axioms for defining *obligations*.

Summarising, we provide: an axiomatic definition of a *generic framework* for the specification of access control and obligation models, obtained by extending the CBAC metamodel with notions of obligation and duty; a rewrite-based *operational semantics* for the extended metamodel, dealing with *authorisation and obligation* assessment, including mechanisms for the resolution of conflicts between authorisations and obligations; and a rewrite-based technique to prove properties of access control policies involving obligations.

Overview: In Section 2, we recall the CBAC metamodel. Section 3 discusses obligations, Section 4 presents the extended CBAC metamodel, and Section 5 the operational semantics for obligations. In Section 6, we discuss related work, and in Section 7, conclusions are drawn and further work is suggested.

2 Preliminaries

We assume familiarity with basic notions on first-order logic and term-rewriting systems [2]. We briefly describe below the key concepts underlying the category-based metamodel of access control; see [3] for a detailed description.

Informally, a category is any of several distinct classes or groups to which entities may be assigned. Entities are denoted by constants in a many sorted domain of discourse, including: a countable set $\mathcal{C}$ of categories, denoted $c_0, c_1, \ldots$, a countable set $\mathcal{P}$ of principals, denoted $p_0, p_1, \ldots$ (we assume that principals that request access to resources are pre-authenticated), a countable set $\mathcal{A}$ of *actions*, denoted $a_0, a_1, \ldots$, a countable set $\mathcal{R}$ of *resources*, denoted $r_0, r_1, \ldots$, a finite set $\mathcal{A}uth$ of possible *answers* to access requests (e.g., $\{\mathsf{grant}, \mathsf{deny}, \mathsf{undetermined}\}$) and a countable set $\mathcal{S}$ of *situational identifiers* to denote environmental information. More generally, entities can be represented by a data structure (e.g., a principal could be represented by a term $\mathit{principal}(p_i, \mathit{attributeList})$), but constants will be sufficient for most examples in this paper. A *permission* is a pair (a, r) of an action and a resource, and an *authorisation* is a triple (p, a, r) that associates a permission with a principal. The metamodel includes the following relations to formalise these notions:

- *Principal-category assignment:* $\mathcal{PCA} \subseteq \mathcal{P} \times \mathcal{C}$, such that $(p, c) \in \mathcal{PCA}$ iff a principal $p \in \mathcal{P}$ is assigned to the category $c \in \mathcal{C}$.
- *Permission-category assignment:* $\mathcal{ARCA} \subseteq \mathcal{A} \times \mathcal{R} \times \mathcal{C}$, such that $(a, r, c) \in \mathcal{ARCA}$ iff action $a \in \mathcal{A}$ on resource $r \in \mathcal{R}$ can be performed by the principals assigned to the category $c \in \mathcal{C}$.
- *Authorisations:* $\mathcal{PAR} \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{R}$, such that $(p, a, r) \in \mathcal{PAR}$ iff a principal $p \in \mathcal{P}$ can perform the action $a \in \mathcal{A}$ on the resource $r \in \mathcal{R}$.

Definition 1 (Axioms). *The relation $\mathcal{PAR}$ satisfies the following core axiom, where we assume that there exists a relationship $\subseteq$ between categories; this can simply be equality, set inclusion (the set of principals assigned to $c \in \mathcal{C}$ is a subset of the set of principals assigned to $c' \in \mathcal{C}$), or an application specific relation.*

$$(a1)\ \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R},$$
$$(\exists c, c' \in \mathcal{C}, ((p, c) \in \mathcal{PCA} \wedge\ c \subseteq c'\ \wedge (a, r, c') \in \mathcal{ARCA}) \Leftrightarrow (p, a, r) \in \mathcal{PAR})$$

Operationally, axiom $(a1)$ can be realised through a set of functions, as shown in [5]. We recall the definition of the function $\mathsf{par}(P, A, R)$ below; it relies on functions pca, which returns the list of categories assigned to a principal, and arca, which returns a list of permissions assigned to a category.

$$(a2)\ \mathsf{par}(P, A, R) \rightarrow if\ \ (A, R) \in \mathsf{arca}^*(\mathsf{below}(\mathsf{pca}(P)))\ then\ \mathsf{grant}\ else\ \mathsf{deny}$$

The function below computes the set of categories that are subordinate to any of the categories in the list $\mathsf{pca}(P)$. The function $\in$ is a membership operator on lists, grant and deny are answers, and arca^* generalises the function arca to take into account lists of categories.

The axiom $(a1)$, and its algebraic version $(a2)$, state that a request by a principal p to perform the action a on a resource r is authorised if p belongs to a category c such that for some category below c (e.g., c itself) the action a is authorised on r, otherwise the request is denied. There are other alternatives, e.g., considering *undeterminate* as answer if there is not enough information to grant the request. More generally, the relations $\mathcal{BARCA}$ and $\mathcal{BAR}$ were introduced in [6] to explicitly represent prohibitions, that is, to specify that an action is forbidden on a resource; $\mathcal{UNDET}$ was introduced to specify undeterminate answers. These relations obey the following axioms:

$$(c1)\ \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R},\\ ((\exists c \in \mathcal{C}, \exists c' \in \mathcal{C}, (p,c) \in \mathcal{PCA} \wedge c' \subseteq c\ \wedge (a,r,c') \in \mathcal{BARCA}) \Leftrightarrow\\ (p,a,r) \in \mathcal{BAR})$$

$$(d1)\ \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R},\\ ((p,a,r) \notin \mathcal{PAR}\ \wedge\ (p,a,r) \notin \mathcal{BAR}) \Leftrightarrow (p,a,r) \in \mathcal{UNDET}$$

$$(e1)\ \mathcal{PAR} \cap \mathcal{BAR} = \emptyset$$

3 Obligations and Events

Obligations differ from permissions in the sense that, although permissions can be issued but not used, an obligation usually is associated with some mandatory action, which must be performed at a time defined by some temporal constraints or by the occurrence of an event. Fulfilling the obligations may require certain permissions, which can lead to undesirable interactions between permissions and obligations, where the obligations in a policy cannot be fulfilled given the set of assigned permissions in the system at a certain state. If obligations go unfulfilled (that is, become violated), this raises the question of accountability, that is, to whom shall be imputed responsibility for unfulfilled obligations. To address these issues, we will extend the CBAC metamodel in order to be able to specify the assignment of obligations to principals and study the interaction between obligations and permissions. We will define obligations consisting of an action (an atomic or a composed action) on a resource to be performed during an interval specified by an initial event and a terminal event.

Following [16], we consider events as action occurrences, or action happenings, that is, an *event* represents an action that happened in the system. This notion has been used in previous works [15, 7, 12]; here we distinguish between *event types*, denoted by ge_i (e.g., registration for a course, a fire alarm, etc.) and specific events, denoted by e_i (e.g., the student *Max M.* registered for the Logic course in September 2012, the fire alarm went off in the Strand Building at 5pm on the 30 June 2012). A typing function will be used to classify events.

Definition 2 (Event History and Interval). *An event history, denoted by $h \in \mathcal{H}$, is a sequence of events that may happen in a run of the system being modelled. A subsequence i of h is called an* event interval*; the first event in i* opens *the interval and the last one* closes *it.*

We assume that an event history contains all the relevant events in order to determine, at each point, the set of authorisations and obligations of each principal (i.e., to determine the system state). Events and event types will be specified using a special-purpose language described in Section 4.2. We assume that the typing relation associating event types with events is decidable.

Definition 3 (Obligation). *A* generic obligation *is a tuple (a, r, ge_1, ge_2), where a is an action, r a resource, and ge_1, ge_2 two event types (ge_1 triggers the obligation, and ge_2 ends it). If there is no starting event (resp., no ending event) we write $(a, r, \perp, ge)$ (resp., $(a, r, ge, \perp)$), meaning that the action on the resource must be performed at any point before an event of type ge (resp. at any point after an event of type ge).*

Example 1. Assume that in an organisation, the members of the security team must call the fire-department if a fire alarm is activated, and this must be done before they de-activate the alarm. This obligation could be represented by the tuple $(call, firedept, alarmON, alarmOFF)$;

Models of access control specify *authorisations* by defining the way *permissions* are assigned to principals. Similarly, an obligation model should specify the way *obligations* are assigned to principals, and it should be possible to determine, at each point in the history, which obligations assigned to principals have to be enforced. For the latter, we introduce the notion of a *duty.*

Definition 4 (Duty). *A* duty *is a tuple (p, a, r, e_1, e_2, h), where p is a principal, a an action, r a resource, e_1, e_2 are two events and h is an event history that includes an interval opened by e_1 and closed by e_2. We replace e_1 (resp. e_2) with $\perp$ if there is no starting (resp. closing) event.*

Unlike access control models, which do not need to check whether the authorised actions are performed or not by the principals, obligation models need to include mechanisms to check whether duties were discharged or not. Specifically, obligation models distinguish four possible states for duties: *invalid* (when at the point a duty is issued the completion point has already passed); *fulfilled* (when the obligation is carried out within the associated interval); *violated* (when it is not carried out within the associated interval) and *pending* (when the obligation has not yet been carried, but the interval is still valid). In some cases, p's duty can be fulfilled by another principal. This is the case in Example 1 above, where all members of the security team have the obligation to call the fire department before deactivating the alarm, but the obligation is fulfilled as soon as one of them makes the call. In order to distinguish both kinds of obligations, we will call *individual obligations* those that have to be fulfilled necessarily by the principal to whom the obligation is assigned, and *collective obligations* those where the obligation is assigned to several principals and can be fulfilled by any of the principals in the group.

4 Obligations in the category-based metamodel

The notion of a category will be used to specify obligations that apply to groups of principals. However, the groupings of principals for authorisations and for obligations are not necessarily the same (for instance, the category UG student is assigned a certain set of permissions, which all UG students enjoy, whereas some UG students belong to the home student category and others to the international student category, with different obligations). We call *permission categories* those used to specify authorisations, and *obligation categories* those used to specify duties. To model obligations and duties we extend the metamodel to include the following sets of entities and relations in addition to the ones defined in section 2:

- Countable sets $\mathcal{E}$ and $\mathcal{GE}$ of events and event types, denoted by $e, e_0, e_1, \ldots$ and $ge, ge_0, ge_1, \ldots$, respectively.
- A countable set $\mathcal{H}$ of event histories, denoted by $h, h_0, h_1, \ldots$.
- *Obligation-category assignment:* $\mathcal{OARCA} \subseteq \mathcal{A} \times \mathcal{R} \times \mathcal{GE} \times \mathcal{GE} \times \mathcal{C}$, such that $(a, r, ge_1, ge_2, c) \in \mathcal{OARCA}$ iff the action $a \in \mathcal{A}$ on resource $r \in \mathcal{R}$ must be performed in the interval defined by two events of type ge_1, ge_2 by principals assigned to the category $c \in \mathcal{C}$. To accommodate individual and collective obligation assignments, this relation is partitioned into two relations: $\mathcal{OARCA}_{\mathcal{I}}$ and $\mathcal{OARCA}_{\mathcal{C}}$. Thus, $(a, r, ge_1, ge_2, c) \in \mathcal{OARCA}_{\mathcal{I}}$ if *every* member of the category c must perform the action $a \in \mathcal{A}$ on resource $r \in \mathcal{R}$ in the interval defined by ge_1, ge_2, and $(a, r, ge_1, ge_2, c) \in \mathcal{OARCA}_{\mathcal{C}}$ if *any* member of c must perform the action $a \in \mathcal{A}$ on resource $r \in \mathcal{R}$ in the interval defined by ge_1, ge_2 and it is sufficient that *one* of them does it.
- *Obligation-principal assignment:* $\mathcal{OPAR} \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{R} \times \mathcal{GE} \times \mathcal{GE}$, such that $(p, a, r, ge_1, ge_2) \in \mathcal{OPAR}$ iff a principal $p \in \mathcal{P}$ must perform the action $a \in \mathcal{A}$ on the resource $r \in \mathcal{R}$ in the interval defined by two events of type ge_1, ge_2. If individual and collective obligations are modelled, then this relation is

replaced by two relations: $\mathcal{OPAR}_I \subseteq \mathcal{P}\times\mathcal{A}\times\mathcal{R}\times\mathcal{GE}\times\mathcal{GE}$, defining individual obligations, and $\mathcal{OPAR}_C \subseteq \mathcal{P}\times\mathcal{A}\times\mathcal{R}\times\mathcal{GE}\times\mathcal{GE}\times\mathcal{C}$ for collective obligations, such that $(p, a, r, ge_1, ge_2) \in \mathcal{OPAR}_\mathcal{I}$ iff principal $p \in \mathcal{P}$ must perform the action $a \in \mathcal{A}$ on the resource $r \in \mathcal{R}$ in the interval defined by two events of type ge_1, ge_2, and $(p, a, r, ge_1, ge_2, c) \in \mathcal{OPAR}_\mathcal{C}$ if principal $p \in \mathcal{P}$ or any other member of c must perform the action $a \in \mathcal{A}$ on the resource $r \in \mathcal{R}$ in the interval defined by two events of type ge_1, ge_2.

- *Duty:* $\mathcal{DPAR} \subseteq \mathcal{P}\times\mathcal{A}\times\mathcal{R}\times\mathcal{E}\times\mathcal{E}\times\mathcal{H}$, such that $(p, a, r, e_1, e_2, h) \in \mathcal{DPAR}$ iff a principal $p \in \mathcal{P}$ must perform the action $a \in \mathcal{A}$ on the resource $r \in \mathcal{R}$ in the interval between the events e_1 and e_2 in the event history h.
  To accommodate individual and collective obligations, this relation is partitioned into $\mathcal{DPAR}_I$ and $\mathcal{DPAR}_C$, similarly to $\mathcal{OPAR}$.
- *Event Typing*: $\mathcal{ET} \subseteq \mathcal{E}\times\mathcal{GE}\times\mathcal{H}$, such that $(e, ge, h) \in \mathcal{ET}$ if the event e is an instance of ge in h. This will be abbreviated as $h \vdash e : ge$.
- *Event Interval*: $\mathcal{EI} \subseteq \mathcal{E}\times\mathcal{E}\times\mathcal{H}$, such that $(e_1, e_2, h) \in \mathcal{EI}$ if the event e_2 closes the interval started by the event e_1 in h.

4.1 Obligation Axioms

In the metamodel, the relations $\mathcal{OPAR}$ and $\mathcal{DPAR}$ are derivable from the others. They satisfy the following core axioms, where we assume that there exists a relationship $\subseteq_o$ between obligation categories; this can simply be equality, set inclusion (the set of principals assigned to $c \in \mathcal{C}$ is a subset of the set of principals assigned to $c' \in \mathcal{C}$), or an application specific relation may be used.

$$
\begin{aligned}
&(o1)\ \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R}, \forall ge_1, ge_2 \in \mathcal{GE},\\
&\qquad \big((\exists c, c' \in \mathcal{C}, (p, c) \in \mathcal{PCA} \land\ c \subseteq_o c'\ \land (a, r, ge_1, ge_2, c') \in \mathcal{OARCA})\\
&\qquad\qquad\qquad\qquad \Leftrightarrow (p, a, r, ge_1, ge_2) \in \mathcal{OPAR}\big)\\
&(o2)\ \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R}, \forall e_1, e_2 \in \mathcal{E}, \forall h \in \mathcal{H}\\
&\qquad\qquad \big((\exists ge_1, ge_2 \in \mathcal{GE}, (p, a, r, ge_1, ge_2) \in \mathcal{OPAR},\\
&\qquad\qquad\quad h \vdash e_1 : ge_1, h \vdash e_2 : ge_2, (e_1, e_2, h) \in \mathcal{EI})\\
&\qquad\qquad\qquad \Leftrightarrow (p, a, r, e_1, e_2, h) \in \mathcal{DPAR}\big)
\end{aligned}
$$

The axiom $o1$ is the essence of the category-based metamodel: it specifies that the principals that are members of a category to which the obligation (a, r, ge_1, ge_2) has been assigned have this obligation. The axiom $o2$ shows how to derive duties. The relation $\subseteq_o$ specifies a hierarchy between obligation categories, which does not necessarily correspond to the way permissions are inherited (specified by the relation $\subseteq$ in axiom $(a1)$).

To accommodate collective and individual obligations, the following variants of the axioms should be included.

$$\begin{aligned}
&(o1_I)\ \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R}, \forall ge_1, ge_2 \in \mathcal{GE},\\
&\qquad \big(\big(\exists c, c' \in \mathcal{C}, (p,c) \in \mathcal{PCA} \wedge\ c \subseteq_o c'\ \wedge (a, r, ge_1, ge_2, c') \in \mathcal{OARCA_I}\big)\\
&\qquad\qquad \Leftrightarrow (p, a, r, ge_1, ge_2) \in \mathcal{OPAR_I}\big)\\
&(o1_C)\ \forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R}, \forall ge_1, ge_2 \in \mathcal{GE}, \forall c' \in \mathcal{C},\\
&\qquad \big(\big(\exists c \in \mathcal{C}, (p,c) \in \mathcal{PCA} \wedge\ c \subseteq_o c'\ \wedge (a, r, ge_1, ge_2, c') \in \mathcal{OARCA_C}\big)\\
&\qquad\qquad \Leftrightarrow (p, a, r, ge_1, ge_2, c') \in \mathcal{OPAR_C}\big)\\
&(o1')\ \ \mathcal{OPAR_I} \cap \overline{\mathcal{OPAR_C}} = \emptyset
\end{aligned}$$

where $\overline{\mathcal{OPAR_C}}$ represents the projection of the relation which discards the last column (the category). Variants of the axiom $o2$ are defined similarly. Axiom $(o1')$ indicates that the same obligation cannot be both collectivelly and individually fulfilled. Additionally, the relation $\mathcal{OARCA_C}$ should not assign the same obligation to two categories in the $\subseteq_o$ relation, to avoid redundancy.

4.2 Event and event type representation

To consider examples of obligation policies we will describe a possible representation for events and event types.

Actions can be either *elementary* or *compound*, i.e. consisting of sets of (simultaneously happened) elementary actions [12]. We use letters a and e, possibly with indexes, to identify actions and events, respectively. We will use a binary representation for events, introduced in [4] and partly motivated by Davidson's work on event semantics [10]. This choice provides a most flexible representation with negligible computational overheads. In our case, the set of arguments depends not only on the action involved but also on the context where the event description is used.

Definition 5 (Event). *An* event description *is a finite set of ground 2-place facts (atoms) that describe an event, uniquely identified by* $e_i, i \in \mathbb{N}$*, and which includes two necessary facts, written* $happens(e_i, t_j)$ *and* $act(e_i, a_l)$*, and* n *non-necessary facts (*$n \geq 0$*).*

In [4] an event description includes three *necessary* facts, and n non-necessary facts ($n \geq 0$). Unlike [4], we use only two types of necessary facts. Their intended meanings may be described as follows: $happens(e_i, t_j)$ means the event e_i happens at time t_j; $act(e_i, a_l)$ means the event e_i involves an action a_l[1]. The events should be positioned in the history in the order in which they happened.

Sometimes we will consider only the predicate *act*, omitting *happens*. In this case, the history would include both events with specified time and events

[1] We restrict attention to events without a duration.

without time, but whether we use the *happens* predicate or not, the history should reflect the order in which the events happened in the system.

If the event e_i involves the subject s_m, then the corresponding non-necessary fact $subject(e_i, s_m)$ can be added to the event description when we need this fact. In [4], $subject(e_i, s_m)$, would be the third necessary fact. Similarly, if the event e_i involves the resource r_l, then the fact $object(e_i, r_l)$ can be added to the description. And so on. Thus, the event description given by the set $\{happens(e_i, t_j)$, $act(e_i, a_l)$, $subject(e_i, s_m)$, $object(e_i, r_l)\}$ represents a happening e_i at a time t_j of an action a_l performed by a subject s_m on a resource r_l.

To define obligations associated with events, here we also consider event types, denoted by ge_i. As it was noted earlier, in real modelling the set of non-necessary predicates involved in a description of an event is determined not only by the action assigned to this event but also by current context. We define an *event type*, or *generic event*, to consist of the necessary fact indicating an action and a set of predicates over expressions with variables including another necessary predicate $happens/2$.

Definition 6 (Event type and Instance). *A set ge of binary atoms represents an event type, also called* generic event*, if there exists a substitution σ such that the instantiation of ge with σ, written, $ge\sigma$ is an event description (see Definition 5).*

An event e is an instance *of a generic event ge, denoted as $e : ge$, if there is a substitution σ such that $ge\sigma \subseteq e$. In other words, if ge subsumes e.*

The action of a substitution σ on a generic event ge may simply be a syntactic operation (replacing variables by terms), or, depending on the application and the kind of data used to define events, instantiation may require some computation; we call it a *semantic instantiation* in the latter case. We give examples below.

Example 2. The events

$$e_1 = \{happens(e_1, 12.25), act(e_1, activate), object(e_1, alarm)\}$$
$$e_2 = \{happens(e_2, 12.45), act(e_2, deactivate), object(e_2, alarm), subject(e_2, peter)\}$$

are instances, with respective substitutions $\sigma_1 = \{E \mapsto e_1, T \mapsto 12.25\}$ and $\sigma_2 = \{E \mapsto e_2, T \mapsto 12.25, X \mapsto peter\}$, of the generic events

$$\begin{aligned} alarmON &= \{happens(E, T), act(E, activate), object(E, alarm)\} \\ alarmOFF &= \{happens(E, T + 20), act(E, deactivate), object(E, alarm), \\ &\quad subject(E, X)\} \end{aligned}$$

In the case of e_2 we are using a semantic instantiation function.

Example 3. Assume that in a given university, every international student must have a valid visa before registration day (when documents are checked). We can define a generic event:

$$registration\text{-}day = \{happens(E, T), act(E, register), subject(E, secretary)\}$$

and define categories "home-student" and "international-student", such that $(obtain, visa, \bot, registration\text{-}day, international\text{-}student) \in \mathcal{OARCA}$[2]. Here the event type that initiates the obligation is not specified ($\bot$), but the final one is ($registration\text{-}day$). If attendance is required for all students, then $\mathcal{OARCA}$ should contain the tuples:

$$(attend, reg\text{-}office, registration\text{-}day\text{-}start, registration\text{-}day\text{-}end, home\text{-}student)$$
$$(attend, reg\text{-}office, registration\text{-}day\text{-}start, registration\text{-}day\text{-}end, international\text{-}student)$$

Example 4. Consider a hospital, where doctors are allowed to read and write the medical records of the patients in their department. Assume that the policy in place also states that any doctor is allowed to read the medical record of a patient who is in an emergency situation (even if the patient is not in their department) but in that case they or any doctor in their department must declare this access (by creating an entry in the hospital log). Let *patient* be a category consisting of all patients (of a given hospital), and *doctor* be a category consisting of all doctors (of the given hospital). Let $patient(X)$ be a (parameterised) category consisting of all patients in the department X, and $doctor(X)$ be a (parameterised) category consisting of all doctors in the department X, such that for all X, $doctor(X) \subseteq doctor$, i.e., the category $doctor(X)$ inherits all permissions from the category *doctor* and similarly $patient(X) \subseteq patient$. Assume that these categories and hierarchical relation are used both for permissions and for obligations.

To model this scenario, we assume the relations $\mathcal{PCA}$ and $\mathcal{ARCA}$ satisfy the following axioms, where $emerg(bcrd, P)$ is true if an event *brcd* initiating a cardiac emergency for P has been detected, and no event ending the emergency has been recorded:

$$\forall P, \forall D,\ (P, patient(D)) \in \mathcal{PCA} \ \Rightarrow\ (read, record(P), doctor(D)) \in \mathcal{ARCA}$$
$$\forall P, \forall D,\ (P, patient(D)) \in \mathcal{PCA} \ \wedge\ emerg(bcrd, P) \Rightarrow$$
$$(read, record(P), doctor) \in \mathcal{ARCA}$$

Moreover, we include the following axiom for $\mathcal{OARCA}_{\mathcal{C}}$, where $gen\_read(X, P)$ is a generic event representing the act of $doctor(X)$ reading the medical record of $patient(P)$:

$$\forall P, \forall X, \forall D, \forall D',\ \big(((P, patient(D)) \in \mathcal{PCA} \wedge (X, doctor(D')) \in \mathcal{PCA} \wedge$$
$$(X, doctor(D)) \notin \mathcal{PCA}) \Rightarrow$$
$$(declare, admin\text{-}log, gen\text{-}read(X, P), \bot, doctor(D')) \in \mathcal{OARCA}_{\mathcal{C}}\big)$$

[2] or $\mathcal{OARCA}_I$ if we need to distinguish between individual and collective obligations.

5 A Rewrite Semantics for Obligations

Operationally, the axioms $o1$ and $o2$ given in Section 4 can be realised through a set of function definitions. In this section we assume all the obligations are individual; the extension to accommodate individual and collective obligations is straightforward. The information contained in the relations $\mathcal{PCA}$ and $\mathcal{OARCA}$ is modelled by the functions pca and oarca, respectively, where pca returns the list of all the categories assigned to a principal and oarca returns the list of obligations assigned to a category, e.g., defined by the rule:

$$\mathsf{oarca}(\mathsf{c}) \to [(\mathsf{a_1}, \mathsf{r_1}, \mathsf{ge_1}, \mathsf{ge'_1}), \ldots, (\mathsf{a_n}, \mathsf{r_n}, \mathsf{ge_n}, \mathsf{ge'_n})].$$

We assume that the function oarca^* takes as input a list of obligation categories and computes the list of obligations for all the categories in the list (similar to arca^*, see Section 2). The function pca was already mentioned in Section 2 for authorisations; for efficiency reasons, a separate function opca could be defined to compute obligation categories.

In addition, we assume that the functions type and $\mathsf{interval}$, specific to the particular system modelled, are also available. The function type implements the typing relation $\mathcal{ET}$ for events, that is, it computes the event type for a given event e in h (taking into account the semantic instantiation relation associated with the events of interest). The function $\mathsf{interval}$ implements the relation $\mathcal{EI}$, which links an event e_1 with the event e_2 that closes the interval started by e_1 in h.

Definition 7. *The rewrite-based specification of the axiom* $(o1)$ *in section* 4.1 *is given by the rewrite rule:*

$$(r1)\ \mathsf{opar}(p, a, r, ge_1, ge_2) \to (a, r, ge_1, ge_2) \in \mathsf{oarca}^*(\mathsf{obelow}(\mathsf{opca}(p)))$$

where the function $\in$ *is a membership operator on lists, and, as the function name suggests,* obelow *computes the set of categories that are below (w.r.t. the hierarchy defined by the* $\subseteq_o$ *relation) any of the categories given in the list* $\mathsf{opca}(p)$*. For example, for a given category* c*, this could be achieved by using a rewrite rule* $\mathsf{obelow}([\mathsf{c}]) \to [\mathsf{c}, \mathsf{c_1}, \ldots, \mathsf{c_n}]$*. Intuitively, this means that if* c' *is below* c*, then* c *inherits all the obligations of* c'*.*

The rewrite-based specification of the axiom $(o2)$ *is given by:*

$$(r2)\ \mathsf{duty}(p, a, r, e_1, e_2, h) \to \mathsf{opar}(p, a, r, \mathsf{type}(e_1, h), \mathsf{type}(e_2, h))\ \textit{and}\ \mathsf{interval}(e_1, e_2, h)$$

where the functions type *and* $\mathsf{interval}$ *are specific to the system modelled, as mentioned above. Additionally, we consider the following function to check the* *status of obligations for a principal p with respect to a history of events:*

$$(r3)\ \textit{eval-obligation}(p, a, r, ge_1, ge_2, h) \rightarrow if\ \mathsf{opar}(p, a, r, ge_1, ge_2)\ then$$
$$\mathsf{append}(\textit{chk-closed}^*(\textit{closed}(ge_1, ge_2, h), p, a, r), \textit{chk-open}^*(\textit{open}(ge_1, ge_2, h), p, a, r))$$
$$else\ [\textit{not-applicable}]$$

where the function append *is a standard function that concatenates two lists,* closed *computes the sublists of h that start with an event e_1 of type ge_1 and finish with an event e_2 of type ge_2 that closes the interval for this obligation, and similarly* open *returns the subhistories of h that start with the event e_1 of type ge_1 and for which there is no event e_2 of type ge_2 in h that closes the interval for this obligation.*

The function chk-closed *with inputs h', p, a, r checks whether in the subhistory h' there is an event where the principal p has performed the action a on the resource r, returning a result* fulfilled *if that is the case, and* violated *otherwise.*

The function chk-open *with inputs h', p, a, r checks whether in the subhistory h' there is an event where the principal p has performed the action a on the resource r, returning a result* fulfilled *if that is the case, and* pending *otherwise.*

The functions chk-closed* *and* chk-open* *do the same but for each element of a list of subhistories, returning a list of results.*

According to the previous specification, if the obligation (a, r, ge_1, ge_2) does not concern p then eval-obligation(p, a, r, ge_1, ge_2, h) returns *not-applicable*, otherwise, the function eval-obligation returns a list of results containing the status of p in relation to this obligation according to h. Usually, h will be the event history that is relevant to the scenario being modelled. For instance, it could be the full history of events in the system, or we could restrict ourselves to the events in the last year, or the events that happened during a specific interval. In our model, it is not possible for a duty to be invalid (thanks to axiom $(o2)$). If h is sufficiently long and depending on the events it contains, it is possible that at different points the obligation was fulfilled, violated or pending. This is why the function eval-obligation(p, a, r, ge_1, ge_2, h) returns a list of results.

If the system modelled includes collective and individual obligations, then the functions chk-closed and chk-open should take into account this in order to check if the obligation has been fulfilled.

Example 5. Consider again the hospital scenario mentioned in Example 4. Assume an investigation is taking place to establish if Doctor Tom Smith, who treated patient J. Lewis in an emergency situation occurring in November 2012, but is not J. Lewis's doctor, has fulfilled his obligation with respect to declaring the access to this patient's records. This is a collective obligation which can be discharged by any of the doctors in his department. Accordingly, we assume the functions chk-closed and chk-open perform collective checks.

In this case, we simply evaluate the term

$$\text{eval-obligation}(TomSmith, \mathit{declare}, \mathit{admin\text{-}log}, \mathit{gen\text{-}read}, \bot, h)$$

where $\mathit{gen\text{-}read} = \{act(E, \mathit{read\text{-}pr(JLewis)}, sub(E, TomSmith)\}$ and h is the event history that starts on the 1st November 2012 and ends today.

5.1 Analysis of policies

In previous work, rewriting techniques were used to derive properties of authorisation policies in the metamodel (see, e.g., [5]). Here we apply similar techniques to prove properties of policies that include obligations.

Unicity of evaluation. Given a policy specification, eval-obligation(p, a, r, ge_1, ge_2, h) should return *a unique answer*, either indicating that the obligation (a, r, ge_1, ge_2) does not apply to p, or providing the status of the corresponding duty in each of the relevant intervals in h (i.e., compute a list indicating whether it was fulfilled, violated or is still pending in each relevant interval). To prove this property, it is sufficient to prove confluence (which implies the unicity of normal forms) and termination (which implies the existence of normal forms for all terms) of the rewrite system specifying opar, duty and eval-obligation, together with sufficient completeness (which characterises the normal forms). These properties will depend on the specific functions defined for the policy modelled (that is, the specific rules used to compute pca, opca, arca, oarca, etc.), and unfortunately they are generally undecidable. However, sufficient conditions exist and tools such as CiME [9] could be used to help checking properties of the first-order rewrite rules that provide the operational semantics of the policy.

Compatibility. The metamodel is sufficiently expressive to permit the definition of policies where authorisations and obligations co-exist and may depend on each other. In this case, security administrators need to check that obligations and authorisations are compatible: the policy should ensure that every principal has the required permissions in order to fulfill all of its duties. This is not trivial because both the authorisation relation $\mathcal{PAR}$ and the obligations relation $\mathcal{OPAR}$ may be defined using categories whose definition takes into account dynamic conditions (e.g., h). In practice, in order to ensure that only duties that are consistent with authorisations are issued, the semantics could be modified, by adding, as a condition for the assignment of an obligation to a principal in axiom ($o1$), the existence of a corresponding authorisation in $\mathcal{PAR}$. More precisely, consider the relation $\mathcal{OPAR}^+$ defined by adding $(a, r, c) \in \mathcal{PAR}$ to the left-hand side in the axiom ($o1$), i.e. ($o1$) is replaced with ($o1^+$)

$$\begin{aligned}(o1^+)\ &\forall p \in \mathcal{P},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R}, \forall ge_1, ge_2 \in \mathcal{GE},\\ &\big(\exists c, c' \in \mathcal{C}, \big((p, c) \in \mathcal{PCA} \wedge\ c \subseteq_o c'\ \wedge (a, r, ge_1, ge_2, c') \in \mathcal{OARCA}\\ &\qquad\qquad \wedge (p, a, r) \in \mathcal{PAR}\big) \Leftrightarrow (p, a, r, ge_1, ge_2) \in \mathcal{OPAR}^+\big)\end{aligned}$$

In the operational semantics, this would boil down to adding in the right-hand side of the rule $r2$ the condition $\mathsf{par}(p, a, r, h, e_1, e_2)$, where the function par computes the authorisation relation within the interval defined by e_1 and e_2 in h. We prefer to follow the separation of concerns principle [11]: instead of adding a condition in ($o1$) to force the compatibility of obligations and authorisations, we axiomatise authorisations and obligations separately, and then check that the authorisations specified are sufficient to enable all the obligations to be fulfilled. Let us first formalise the notion of compatibility.

Definition 8 (Compatibility). *A policy is* compatible *if*

$$\mathcal{OPAR}^+ = \mathcal{OPAR} \tag{1}$$

A policy is weakly compatible *if*

$$\overline{\mathcal{OARCA}}\ \cap \mathcal{BARCA} = \emptyset \tag{2}$$

where $\overline{\mathcal{OARCA}}$ is the projection of the relation $\mathcal{OARCA}$ on the arguments a, r, c, i.e. in notations of relational algebra $\overline{\mathcal{OARCA}} = \pi_{a,r,c}(\mathcal{OARCA})$.

A policy is strongly compatible *if*

$$\overline{\mathcal{OARCA}}\ \subseteq\ \mathcal{ARCA} \tag{3}$$

Below we summarise the entailment relations between compatibility, strong compatibility and weak compatibility.

Property 1.
1. Strong compatibility implies weak compatibility, ($\mathbf{3} \Rightarrow \mathbf{2}$), under the condition that categories are not empty (i.e., there is at least one principal assigned to each category).
2. Compatibility implies strong compatibility, ($\mathbf{1} \Rightarrow \mathbf{3}$), under the conditions that each principal belongs to a unique category and each category is not empty.
3. Strong compatibility implies compatibility, ($\mathbf{3} \Rightarrow \mathbf{1}$), under the condition that orderings $\subseteq_o$ and $\subseteq$ are the same, more precisely if $\subseteq_o$ implies (is included in) $\subseteq$.

Weak compatibility is, in a sense, minimal among the notions of compatibility defined above since ($\mathbf{2} \not\Rightarrow \mathbf{1}$) and ($\mathbf{2} \not\Rightarrow \mathbf{3}$). Strong compatibility implies weak compatibility, i.e., ($3 \Rightarrow 2$), due to the fact that axiom ($e1$) (see Section 2) implies $\mathcal{ARCA} \cap \mathcal{BARCA} = \emptyset$ if categories are not empty. The proofs of the other implications are omitted due to space constraints. The condition $\subseteq_o$ implies $\subseteq$ means that inheritance of obligations between categories implies inheritance of authorisations. The following sufficient condition for compatibility is a direct consequence of Property 1.

Corollary 1. *A policy is compatible if*

$$\forall c \in \mathcal{C}, \textsf{\textit{subset}}(\textsf{\textit{proj-list}}_{ar}(\textsf{oarca}(c)), \textsf{arca}(c)) \rightarrow^* \textsf{\textit{True}}$$

where we assume $\subseteq_o$ implies $\subseteq$, $\textsf{subset}$ is the function that checks the subset property for sets represented as lists of elements, and $\textsf{proj-list}_{ar}$ is the projection function that projects each element of a list on the first and second components (i.e., action, resource).

Using this result, we can devise a method to automatically prove compatibility of a policy involving authorisations and obligations using a completion tool, such as CiME, to deal with the universal quantification on categories. First, the rewrite system R defining the policy should be proved confluent and terminating by the tool. Then, we add to R the equation $\textsf{subset}(\textsf{proj-list}_{ar}(\textsf{oarca}(C)), \textsf{arca}(C)) = \textsf{True}$. The completion procedure will instantiate C to unify with the left-hand side of the rules defining $\textsf{oarca}$ and $\textsf{arca}$. If the policy is compatible, the completion procedure terminates successfully without adding any rules to the system. If it is not compatible, the completion procedure will detect an inconsistency $\textsf{True} = \textsf{False}$. A similar technique can be used to prove weak compatibility.

6 Related work

The CBAC model we consider here is an extension of the model defined in [5] for authorisations; all the results regarding the rewriting semantics of authorisations are also valid here, but are not the focus of this paper. Several models dealing with the notion of obligations have been proposed in the literature (see, for example, [21, 12, 13, 18, 17, 14, 8]), with different features and providing different levels of abstraction in the definition and management of obligations. Some models consider obligations to be fulfilled by the system and therefore never violated, whereas others consider obligations to be fulfilled by users, in which case other questions arise such as how can the system guarantee that the user will be allowed to fulfill the obligations, and the notion of accountability (if a user, with the necessary permissions, does not fulfill its obligations then he becomes accountable).

The framework in [17] deals both with system and user obligations and measures to enforce the fulfillment of obligations. Obligations are triggered by time or events and violated obligations are treated by the system. Our metamodel includes an abstract notion of a principal, which can be a user or the system, and we distinguish between obligations, which are associated with generic events, and duties, which are triggered by events that happen in the system. Time-triggered obligations and duties can be accommodated by defining events corresponding to clock ticks.

The system presented in [14] extends the usage control (UCON) model with obligations and separates system obligations (which are called trusted obligations) from user obligations (called non-trusted). Mechanisms are proposed to deal with the non-trusted obligations. The system does not consider the interaction of obligations with permissions, neither deals with dynamic conditions on obligations, as our metamodel does. In [20], the ABC core model was defined for UCON, dealing with authorisations, obligations and conditions. This approach differs from ours, since it describes not a general metamodel, but instead a family of models depending on the type of authorisations, obligations and conditions that are considered and the necessary components for each model in the family. Our approach also considers authorisations, obligations and conditions, but in a uniform way, and provides a rewrite-based operational semantics for authorisations and obligations.

The approach followed in [8] was to formalise policies (using Horn clauses) trying to minimise the number of actions that need to be performed before allowing an access request. Although the initial system only dealt with systems without violated obligations, this was later extended to handle violation of obligations. This approach was limited in the type of obligations that could be modelled, because it lacked mechanisms to deal with temporal constraints.

In [13] a model is presented for authorisation systems dealing with the notion of accountability; obligations are defined as an action on a resource by a user within a time interval (defined in clock ticks from a predetermined starting point). The monitoring and restoring of accountability was further explored in [21], where a transition relation is defined and conditions to guarantee accountability are established. The notion of obligation defined in these works corresponds to concrete obligations (duties) in our model, and although this is not the focus of this paper, we believe that rewriting can be used to verify the properties of accountability studied in these papers.

A more general model dealing with obligations and its relation to access control and privacy policies was defined in [18]. This model investigates the interaction of obligations and permissions and its undesirable effects such as obligation cascading. We do not deal with privacy policies here, but the category-based model that we present is expressive enough to model the concepts of obligations defined in this work, and rewriting properties can be used to further explore the interplay between permissions and obligations.

Closely related to our work is [12], which considers obligations and authorisations in dynamic systems using a notion of event defined as a pair of a system state and an action. Our notion of event also includes actions, and system states can be included in the event representation. A major difference is that [12] focuses on compliance of events whereas we focus on compatibility properties of policies. Also, we use a rewriting semantics instead of the logic programming approach advocated in [12]. In this sense, our work and [12] are complementary: an answer set semantics could be defined for our policies, and Prolog used to check compatibility provided a tool is available to check termination of the program.

7 Conclusions

We augmented the CBAC metamodel with a general notion of obligation. The framework is expressive enough to deal with most of the features relevant to authorisations and obligations and provides means to reason about them.

Our model distinguishes individual and collective obligations by partitioning the set of obligations into two distinct sets. However, there are some situations where it can be difficult to distinguish between one or the other type. An alternative approach to be investigated in the future is to base this distinction on the definition of the obligation-category assignment relation. That is, although at a certain moment all the principals belonging to a particular category are obliged to perform some action, the definition of the category can depend on the fact of the action not having been performed at the time. In future work we will further develop the notion of event type, and give an operational definition of the typing relation for events. We also wish to explore appropriate mechanisms to deal with compliance and accountability.

References

1. ANSI. RBAC, 2004. INCITS 359-2004.
2. F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, Great Britain, 1998.
3. S. Barker. The next 700 access control models or a unifying meta-model? In *Proc. of SACMAT 2009*, pages 187–196. ACM Press, 2009.

4. S. Barker, M. J. Sergot, and D. Wijesekera. Status-based access control. *ACM Trans. Inf. Syst. Secur.*, 12(1), 2008.
5. C. Bertolissi and M. Fernández. Category-based authorisation models: operational semantics and expressive power. In *Proc. of ESSOS 2010*, number 5965 in LNCS, pages 140–156. Springer, 2010.
6. C. Bertolissi and M. Fernández. Rewrite specifications of access control policies in distributed environments. In *Proc. of STM 2010*, number 6710 in LNCS. Springer, 2011.
7. C. Bertolissi, M. Fernández, and S. Barker. Dynamic event-based access control as term rewriting. In *Data and Applications Security XXI. Proceedings of DBSEC 2007*, number 4602 in Lecture Notes in Computer Science. Springer-Verlag, 2007.
8. C. Bettini, S. Jajodia, X. Wang, and D. Wijesekera. Provisions and obligations in policy rule management. *Journal of Network and Systems Management*, 11(3):351–372, 2003.
9. E. Contejean, A. Paskevich, X. Urbain, P. Courtieu, O. . Pons, and J. Forest. A3pat, an approach for certified automated termination proofs. In *Proc. of PEPM'10*, pages 63–72, New York, NY, USA, 2010. ACM.
10. D. Davidson. *Essays on Actions and Events.* Oxford University Press, 2001.
11. E. W. Dijkstra. *Selected writings on computing - a personal perspective.* Texts and monographs in computer science. Springer, 1982.
12. M. Gelfond and J. Lobo. Authorization and obligation policies in dynamic systems. In *ICLP*, pages 22–36, 2008.
13. K. Irwin, T. Yu, and W. H. Winsborough. On the modeling and analysis of obligations. In *Proc. of CCS'06*, pages 134–143, New York, NY, USA, 2006. ACM.
14. B. Katt, X. Zhang, R. Breu, M. Hafner, and J.-P. Seifert. A general obligation model and continuity: Enhanced policy enforcement engine for usage control. In *Proc. of SACMAT '08*, pages 123–132, New York, NY, USA, 2008. ACM.
15. R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.
16. R. Miller and M. Shanahan. The event calculus in classical logic - alternative axiomatisations. *Electron. Trans. Artif. Intell.*, 3(A):77–105, 1999.
17. M. C. Mont and F. Beato. On parametric obligation policies: Enabling privacy-aware information lifecycle management in enterprises. In *POLICY*, pages 51–55, 2007.
18. Q. Ni, E. Bertino, and J. Lobo. An obligation model bridging access control policies and privacy policies. In *Proc. of SACMAT'08*, pages 133–142, New York, NY, USA, 2008. ACM.
19. OASIS. eXtensible Access Control Markup language (XACML), 2003. http://www.oasis-open.org/xacml/docs/.
20. J. Park and R. Sandhu. The ucon abc usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, Feb. 2004.
21. M. Pontual, O. Chowdhury, W. H. Winsborough, T. Yu, and K. Irwin. On the management of user obligations. In *Proc. of SACMAT '11*, pages 175–184, New York, NY, USA, 2011. ACM.