Formalization and Management of Group Obligations

Yehia El Rakaiby, Frédéric Cuppens and Nora Cuppens-Boulahia
TELECOM Institute ; TELECOM-Bretagne, Cesson Sévigné, France
Email: {yehia.elrakaiby, frederic.cuppens, nora.cuppens}@telecom-bretagne.eu

Abstract—The specification of abstract security policies which indirectly apply to system entities (like subjects and objects) through group relations (like roles or domains) has been shown to simplify policy specification, interpretation and analysis. In this paper, we show how the abstraction of subjects, actions and objects in obligation policies using group relations can enhance the expressiveness of obligation policy languages. More precisely, we introduce the notion of group contexts through which the policy designer can choose different interpretations for group relations in obligation security rules enabling him or her to specify obligations representing shared responsibilities such as “All patients must be checked by a doctor” or obligations expressing sets of alternative actions such as “Every customer should pay either in cash or by check”. Management and monitoring requirements of such obligations called group obligations are studied and formalized.

I. INTRODUCTION

Security policies express the deontic concepts of permission, prohibition and obligation and aim to control user-actions in the system. To simplify policy management, most policy languages support the specification of policy rules on the group-level using some group membership relations such as roles [1] and domains [2]. Group relations may also be used to specify security rules on groups of actions or objects which share common security properties. For instance, the concepts of activities and views, introduced by the OrBAC model [3], may be used to specify groups of actions and objects, respectively, in security rules.

An obligation represents a duty or a requirement for some subject to take some action on some object. The importance of obligations and their relevance to security policies have been demonstrated and most current security policy languages [4]–[8] support the expression of some form of obligation policies.

The specification of obligation policies on the group-level enables the expression of obligations which apply to more than one subject, e.g. the obligation “All resident doctors must submit an activity report daily”. It also enables the expression of shared responsibilities, e.g. the obligation “Every day, one of the resident doctors must submit the activity report”. One may say that the use of group relations in the first obligation reduces the number of security rules that need to be defined since the obligation represents separate individual obligations for every resident doctor to submit the activity report. On the other hand, group relations in the second obligation enhances the expressiveness of the obligation language since they enable the expression of a shared responsibility which could not have been expressed without group relations.

Similarly, the specification of groups of actions and objects can enable the specification of alternative obligation fulfillment actions. For example, consider an activity pay which contains the actions \{payCash, payByCheck\} and a view taxes which includes the objects \{incomeTax, propertyTax\}. The obligation “taxpayers must take one action in activity pay on all objects in view taxes” represents a requirement that every taxpayer pays his or her income tax and property taxes either in cash or by check.

In this paper, we study how to enable the expression of shared responsibilities and alternative duties when group relations are supported in the obligation language. Management requirements and possible interpretations of these new obligations, called group obligations, are studied and formalized. To the best of our knowledge, this is the first time that this topic is tackled in research on security policy languages.

The remainder of this paper is structured as follows. Section 2 presents some motivating examples. Section 3 introduces the basic concepts used in the formalization of group obligations. Section 4 presents our obligation policy language. Section 5 discusses the management of group obligations. Section 6 presents an application example. Finally, Section 7 discusses related work and Section 8 concludes the paper.

II. MOTIVATING EXAMPLES

To motivate our work, we consider the following obligation “Everyday, patients who are in stable state must be reevaluated (either discharged or re-admitted) by one of the doctors to which they are assigned before the end of the working day. Additionally, doctors should reevaluate patients before the end of their clinic hours”. In the following, we consider that doctors is a role to which are assigned the doctors \(d_1, d_2, d_3\), reevaluate is an activity which contains the actions (discharge, re-admit), and patients is a view which includes the patients \(p_1, p_2, p_3\).

Consider the following scenario. At the start of the day, the obligation is activated. Assume that at this particular moment, patient \(p_1\) is assigned to doctors \(d_1, d_2\), patient \(p_2\) is assigned to doctors \(d_2, d_3\) and patient \(p_3\) is assigned to
doctor \((d_3)\). An intuitive interpretation of the obligation above is that it requires the following:

\[ r_1 : p_1 \text{ should be reevaluated by one of the doctors } (d_1, d_2), \]
\[ r_2 : p_2 \text{ should be reevaluated by one of the doctors } (d_2, d_3), \]
\[ r_3 : p_3 \text{ should be reevaluated by the doctor } (d_3). \]

Since requirements \(r_1\) and \(r_2\) may be fulfilled by more than one subject, we call them group obligations. Subjects appearing in a group obligation are co-responsible for its fulfillment. Therefore, these subjects are notified of the whole group obligation to coordinate its fulfillment. Evidently, subjects are subsequently notified of updates to their group obligations.

Consider that at noon, doctor \(d_2\) is assigned to patient \(p_3\) and de-assigned from the patient \(p_1\). The previous group obligations were updated as follows:

\[ r_1 : p_1 \text{ should now be reevaluated by the doctor } (d_1), \]
\[ r_2 : p_2 \text{ should be reevaluated by one of the doctors } (d_2, d_3), \]
\[ r_3 : p_3 \text{ can now be reevaluated by one of the doctors } (d_2, d_3). \]

Assume that at 14:00, the doctor \(d_2\) re-admits patient \(p_3\) to the hospital. In this case, \(r_3\) is fulfilled and doctors \(d_2\) and \(d_3\) are notified that it is no longer necessary to reevaluate \(p_3\).

The obligation states that doctors should reevaluate patients before the end of their clinic hours. Assume that the clinic hours of doctors \((d_1, d_2, d_3)\) end at 15:00, 16:00 and 19:00 respectively. Now, if by 15:00, doctor \(d_1\) has not reevaluated \(p_1\), obligation \(r_1\) is violated and sanction policies are activated for doctor \(d_1\). On the other hand, if the clinic hours of doctor \(d_2\) ends at 16:00 before doctor \(d_2\) reevaluates \(p_2\), then the group obligation \(r_2\) is not violated since it can still be fulfilled by \(d_3\). However, \(r_2\) is updated and doctor \(d_3\) is notified that he or she is now the only subject who can fulfill \(r_2\).

Assume that the working day ends at 18:00. In this case, doctor \(d_3\) is required to reevaluate the patient \(p_2\) before 18:00. Therefore, if doctor \(d_3\) reevaluates \(p_2\) before 18:00, the group obligation \(r_2\) is fulfilled. If, on the other hand, \(d_3\) fails to reevaluate \(p_2\), \(r_2\) is violated and doctors \((d_2, d_3)\) are considered co-responsible of its violation. Consequently, sanction policies may be activated for \(d_2\) and \(d_3\).

We argue that there is no equivalent notion of group permissions or prohibitions for access control policies. To justify this claim, consider the access control rule \(R_1\): “One doctor is permitted to access the president’s medical record”. This security rule is under-specification since it does not enable the determination of the exact subject to whom the rule applies. On the other hand, \(R_1\) may aim to specify that, at any time, only one doctor should have access to the president’s medical record. In this case, \(R_1\) specifies a policy constraint and not an access control security rule.

In this paper, we formalize the concepts presented through the example above. In particular, we study the construction, update, fulfillment and violation of group obligations. We also show how sanction policies can be applied when group obligations are violated. Finally, obligations which require a certain number of individual obligation fulfillments, e.g. the obligation “A patient in critical state must be examined by two of the doctors within 15 minutes”, are formalized.

III. BASIC CONCEPTS

A. System Object & State Representation

We consider finite sets of subjects, objects, actions and contexts represented by the sorts \(S, O, A\) and \(C\) respectively. We also consider a finite set of predicate symbols \(Q\) and variables \(V\). Constants and variables are terms of the language. An atom is a formula \(Q(t_1, \ldots, t_n)\) where \(Q\) is a predicate symbol and each of \(t_i\) is a term. Variable free atoms are facts. Facts are called fluents to denote that they can change over time.

We support three types of group relationships in our model. Subjects are empowered into roles using the relation \(\text{Empower}(Subject, Role)\), actions are considered in activities using the relation \(\text{Consider}(Action, Activity)\) and objects are used in views using the relation \(\text{Use}(Object, View)\).

B. The Active Language \(L_{\text{active}}\)

Obligations in our model may assume four states as shown in figure 1. To formalize obligation policy management, we use the language \(L_{\text{active}}\) presented in [9]. \(L_{\text{active}}\) provides a formal characterization of active databases and allows reasoning about change in the state of dynamic systems which results from action occurrences in the system. A translation of the language to logical programs is presented in [10].

The alphabet of \(L_{\text{active}}\) consists of four sorts: (1) Fluents: are time-variation propositions or facts representing the system state. (2) Actions: represent possible actions in the system. Actions update the state by adding and removing fluents to and from the state. (3) Events: are used to specify state conditions at which some policy management operations are required. (4) Rule Names: unique identifiers of Event Condition Action (ECA) rules. ECA rules, also called active rules, are used to initiate policy management operations after the detection of events. An ECA rule states that when event occurs and if conditions are true, then actions are executed.

The semantics of \(L_{\text{active}}\) is given by the following three propositions:

\( \text{(EL)} \ a(X) \text{ causes } f(Y) \text{ if } p_1(X_1), \ldots, p_n(X_n) \)
\( \text{(ED)} \ e(Y) \text{ after } a(X) \text{ if } p_1(X_1), \ldots, p_n(X_n) \)
\( \text{(AR)} \ r(X_1) : e(X) \text{ initiates } [a] \text{ if } p_1(X_1), \ldots, p_n(X_n) \)

Where the symbols \(f, p_1, \ldots, p_n\) are fluent symbols, \(a\) is an action symbol, \(e_1, \ldots, e_n\) are event symbols and \(r\) is an active rule identifier. An effect law proposition (EL) states that the execution of \(a(X)\) in a state where the fluents \(p_1(X_1), \ldots, p_n(X_n)\) are true causes \(f(Y)\) to be true in the next state. An event definition proposition (ED) states that
if the conditions \(p_1(X_1), ..., p_n(X_n)\) are true in the state following the execution of the action \(a(X)\), then event \(e(Y)\) is produced. An active rule proposition \((A,R)\) states that every new detection of the event \(e(X)\) initiates the execution of the sequence of actions \([a]\) if the rule conditions are true.

The operational semantics of the language 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, its effect laws are evaluated and the 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. The identifiers of the triggered rules are added to the triggered rules set. When the last action in the sequence is evaluated, if the triggered rules set is not empty, an action selection function selects the sequence of actions appearing in one of the rules to process. As suggested in [9], active rules may be assigned priorities. Thus, our action selection function returns the sequence of actions appearing in one of the rules having the highest priority in the set. The state stops evolving after the processing of all the actions in a sequence if the triggered rule set is empty.

IV. OBLIGATION POLICY LANGUAGE

An individual obligation is a requirement that some action be taken by some subject on some object. Since generally users can not be forced to take actions, obligations may be violated. In our model, an obligation is associated with a set of conditions called the obligation context. We say that an obligation holds (or is active) when these conditions are true. An obligation is also associated with another set of conditions called the obligation violation context. When this context becomes true, the obligation is considered violated if the obligation holds. An obligation ceases to hold whenever it is fulfilled. For instance, consider the obligation “In case of emergency, a security guard is required to evacuate all patients of some department within 15 minutes”. This obligation becomes effective when the context emergency starts to be true. The obligation is violated if the required action, namely the evacuation of the patients, is not taken within 15 minutes. This obligation is deactivated if the context emergency ends before the obligation’s deadline is reached. The obligation is removed (ceases to hold) if it is fulfilled.

A. Obligation Expression

Our policy language supports the specification of obligations using simple subjects, actions and objects or on the group level using roles, activities and views. Obligations defined over the relations roles, activities or views are called organizational obligations. We call individual obligations which can be derived from an organizational obligation, when considering that every subject assigned to the obligation’s role is required to take every action considered in the obligation’s activity on every subject used in the obligation’s view, the obligation members.

Obligation policies are closed ground facts of the following form:

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

Where \(N\) is a unique security rule identifier, \(SR\) is a subject or a role, \(AA\) is an action or an activity and \(OV\) is an object or a view. The obligation context \(Ctx\) represents the set of conditions which, while true, the obligation holds. The violation context \(Ctx_v\) represents the set of conditions which, if true while the obligation holds, the obligation is violated. The specification and interpretation of contextual contexts are presented in Section IV-B.

The GroupContext enables the specification of particular relationships between organizational obligation members and is expressed as follows: groupContext(∀qe \(Q\) ∃qe \(Q\) qe \(Q\) qe \(Q\)) where every \(Q\) is one of the quantifiers \(\{\forall, \exists\}\) and each \(qe\) is a different element of \(\{s, a, o\}\). When the GroupContext is not specified, the organizational obligation is associated with the default group context groupContext(vis\(\forall\)a\(\exists\)o).

We say that the interpretation of an organizational obligation determines its group requirements. Organizational obligation interpretation refers to the process by which we derive from the organizational obligation and its group context the obligation requirements. An obligation requirement is a set of one or more alternative individual obligations. The interpretation of organizational obligations is formalized in Section IV-C. For example, consider the organizational obligation “Everyday, patients must be checked by one of the doctors at the hospital before the end of the day” specified as follows:

\[
\text{Obligation}(\forall o, \exists doctors, check\_up, patients, \\
\text{start}(workingHours) \& \text{doctorAtHospital}, \\
\text{end}(workingHours), groupContext(vis\(\forall\)a\(\exists\)o))
\]

The interpretation of this obligation produces requirements stating that for every object (patient), there should exist an action (check\_up) and a subject (doctors), such that this patient is checked up by the doctor.

B. Contextual Conditions Expression

We consider two context types. Persistent contexts which hold over an interval of time and, thus, enable the specification of properties. For instance, the persistent context atHospital may hold for a subject while the subject is at the hospital. The second type is instantaneous contexts which enable the specification of particular time moments or events. For instance, consider the instantaneous context admittedER. This context may be active exactly at the instant a patient is admitted to the emergency room.

An obligation context may be either a persistent or an instantaneous context. When the obligation context is persistent, the obligation is activated when the context is started and the obligation is deactivated if the context is ended while the obligation holds. When the obligation context is instantaneous, the obligation is activated at every detection of the context. Then, the obligation holds forever until it is either fulfilled or violated. When the obligation violation context is
an instantaneous context, the obligation is violated whenever the context is detected while the obligation holds. When the obligation violation context is persistent, the obligation is violated when the context is started.

Security rules specify whether a subject (S) is permitted, prohibited or obliged to take action (A) over object (O). Therefore, they typically include the triple (S,A,O). To associate security rules with contextual conditions over (S,A,O), we use the event Hold,(S,A,O,Ctx). This event denotes that the set of conditions identified by Ctx is true for the subject S, action A and object O.

Instantaneous contexts (events) are specified directly using the event Hold,(S,A,O,Ctx). For instance, the context doctorEnteredER is specified as follows (the prolog symbol _ is interpreted as representing “do not care” condition):

\[
\begin{align*}
    & \text{Hold},(S,_,\text{doctorEnteredER}) \\
    & \text{after } \text{enter}(S,\text{emergencyRoom}) \\
    & \text{if } \text{Empower}(S,\text{doctors})
\end{align*}
\]

This instantaneous context is detected every time a doctor enters the emergency room and is interpreted as denoting that the context doctorEnteredER holds for the doctor S in the state which results from the occurrence of the action enter(S,emergencyRoom).

A persistent context is specified using the special instantaneous contexts start(Ctxp) and end(Ctxp). These instantaneous contexts specify the moments at which the persistent context starts and ends. For instance, the persistent context doctorAtHospital denoting that some doctor is at the hospital is specified as follows.

\[
\begin{align*}
    & \text{Hold},(S,_,\text{start}(	ext{doctorAtHospital})) \\
    & \text{after } \text{enter}(S,\text{hospital}) \text{ if } \text{Empower}(S,\text{doctors})
\end{align*}
\]

\[
\begin{align*}
    & \text{Hold},(S,_,\text{end}(	ext{doctorAtHospital})) \\
    & \text{after } \text{exit}(S,\text{hospital}) \text{ if } \text{Empower}(S,\text{doctors})
\end{align*}
\]

The persistent context doctorAtHospital remains true over the interval which starts from the moment the event context start(doctorAtHospital) is detected until the moment the context end(doctorAtHospital) is detected. The start (end respectively) of a persistent context is only detected if the context starts (ends respectively) while the context is false (true respectively). Our context language also supports the specification of temporal contexts using the special action Clock. This action has attributes representing calendars available in the system, e.g. Time, Day, Week, Month and Year. For instance, the context workingHours may be specified as follows:

\[
\begin{align*}
    & \text{Hold},(S,_,\text{start}(\text{workingHours})) \\
    & \text{after } \text{Clock} \text{ if } \text{Time}(\text{Clock},08:00)
\end{align*}
\]

\[
\begin{align*}
    & \text{Hold},(S,_,\text{end}(\text{workingHours})) \\
    & \text{after } \text{Clock} \text{ if } \text{Time}(\text{Clock},18:00)
\end{align*}
\]

To enable the composition of contexts, we have developed a context language [11] which supports the standard logical operators conjunction (\&), disjunction (\lor) and negation (\neg). Relative temporal deadlines can be specified using the context Delay(Nb.TimeUnit) which is a context that is detected Nb time units after the instantiation of the obligation.

### C. Group Contexts Semantics and Interpretation

An organizational obligation whose group context is the default group context (\forall S,A,O) is called a regular organizational obligation. Interpretation of a regular organizational obligation produces requirements in the form of separate individual obligations for every subject empowered in the obligation’s role to take every action considered in the obligation’s activity on every object used in the obligation’s view.

When the group context contains the quantifier (\exists), the organizational obligation is said to represent a group obligation. The interpretation of organizational group obligations produces requirements which are not generally in the form of separate individual obligations. In particular, it produces a set of requirements where each requirement is a set of alternative individual obligations. This set is called the obligation requirements set. For instance, the interpretation of the obligation “every patient must be checked by one of the doctors” when there are two doctors (d₁,d₂) and a single patient p₁ produces a requirements set containing the single requirement “(d₁ must check p₁) or (d₂ must check p₁)”.

The interpretation of organizational obligations is determined using the algorithm constructGroupObligations shown in figure 2. The algorithm, given the obligation group context definition and its different obligation members, returns a sentence in conjunctive normal form representing the different obligation requirements. This sentence is a conjunction of disjunctions having the form:

\[
\text{OR}_1 \land \ldots \land \text{OR}_n \text{ where each } \text{OR}_i \text{ is a disjunction of individual obligation instances of the form } \text{Obliged}_1 \lor \ldots \lor \text{Obliged}_n.
\]

The different \text{OR}_i represent the obligation requirements and the different \text{Obliged}_j in any \text{OR}_i represent alternative concrete individual obligation. The violation of any of the requirements \text{OR}_i is said to violate the organizational obligation whereas the fulfillment of any of the individual obligations in \text{OR}_i fulfills the requirement \text{OR}_i.

The sentence is constructed as follows. Every obligation instance in the state is considered an obligation expression \text{Obliged}(id, s, a, o, dx, ct, vx). These expressions are joined using conjunctions and disjunctions to produce a sentence that represents the group obligation requirements by considering the semantics of the quantifiers in the group context definition from right to left. Notice that, although we assume in the paper that an individual obligation involves the triple (subject, action, object), we have defined the above algorithm in a general way to enable the interpretation of obligations involving any other number or types of parameters. The algorithm works as follows. At every iteration, if the considered quantifier is associated to the parameter \text{e}_i in the obligation expression \text{s}_n(id, e_1, ... e_{i-1}, e_i, e_{i+1}, ... e_n) then new obligation expressions \text{s}_{n-1}(id, e_1, ... e_{i-1}, e_{i+1}, ... , e_n) are formed by joining all the different sentences \text{s}_n(id, e_1, ... e_{i-1}, e_i, e_{i+1}, ... , e_n)
Obligation policy management consists of changing the state of obligations as the system state evolves. In our framework, policy management is performed using active rules which initiate policy management actions when particular events are detected. In this section, we present the management rules of both regular and group obligations.

V. ORGANIZATIONAL OBLIGATION POLICY MANAGEMENT

A. Instantiation of Obligations

A regular organizational obligation is activated when the start of its context is detected. This activation instantiates the obligation members whose triple \((S, A, O)\) satisfies the context as specified in the following active rule:

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

\[
\text{initiates Insert_Obligation}\left(N, S, A, O, Ctx, \text{Ctx}_x\right)
\]

\[
\text{if Obligation}(N, R, \text{Act}, V, Ctx, \text{Ctx}_x, \text{Ctx}_y, \text{Ctx}_z),
\]

\[
\text{Empower}(S, R), \text{Consider}(A, \text{Act}), \text{Use}(O, V)
\]

The above rule initiates, when the obligation context is started, the action \text{Insert_Obligation} for every organizational obligation member whose triple \((S, A, O)\) satisfies the context. This action adds to the state obligation instances of the form \(\text{Obligated}(N, S, A, O, Ctx, \text{Ctx}_x)\) to denote that the subject \(S\) is required to take the action \(A\) on the object \(O\) while the context \(Ctx\) holds before the context \(\text{Ctx}_x\) is detected.

When a group obligation is activated, one must consider both the context and the interpretation of the obligation. Similarly to regular obligations, the context is relevant because only member obligations whose context holds are activated. The group obligation interpretation, on the other hand, determines the different obligation instances which satisfy the same group obligation requirement. The obligation interpretation is not relevant in the case of regular organizational obligations since it does not construct any relationships between the different obligation instances, i.e. a requirement always corresponds to a single individual obligation instance. Instantiation of group obligations is managed using the following active rule:

\[
\text{instantiate_Obligation}_g : \text{Hold}(S, A, O, \text{start}(Ctx))
\]

\[
\text{initiates Insert_Obligation} \left(N, S, A, O, \text{Ctx}, \text{Ctx}_x\right)
\]

\[
\text{if Obligation}(N, R, \text{Act}, V, \text{Ctx}, \text{Ctx}_x, \text{Ctx}_y),
\]

\[
\text{Empower}(S, R), \text{Consider}(A, \text{Act}), \text{Use}(O, V)
\]

This active rule instantiates the group obligation \(N\) by adding the obligation instances \(\text{Obligated}(N, S, A, O, Ctx, \text{Ctx}_x)\) whose triple \((S, A, O)\) satisfies the context to the state. To construct the obligation requirements after the instantiation of the different group obligation members, the action \text{Insert_Obligation}_g produces the event \text{groupObI}(N)\) at the state following the triggering of the group obligation to denote that the group obligation \(N\) has been activated. This event triggers an active rule which orders the execution of the algorithm \text{constructGroupObligations} presented in figure 2.

\[
\text{constructObligations} : \text{Hold}(\_\_\_, \_\_\_, \text{groupObI}(N))
\]

\[
\text{after Insert_Obligation}_g(N, \_\_\_, \_\_\_, \_\_\_, \_\_\_)\]

\[
\text{constructObligations} : \text{Hold}(\_\_\_, \_\_\_, \text{groupObI}(N))
\]

\[
\text{initiates ConstructGroupObligations}(N)
\]

To ensure that the algorithm is executed only after the instantiation of the different group obligation members, the active rule \text{constructObligations} is given lesser priority than the instantiation rule \text{instantiate_Obligation}_g. After the execution of the algorithm, the group obligation instances are replaced by the group obligation requirements set \text{setGroupObligations}(N).

Subjects are notified of their obligations as follows: if some subject \(s\) appears in some requirement \(OR(N)\), then \(s\) is notified
that he or she is co-responsible for the obligation expressed by $OR(N)$. Subsequently, $i$ is notified of any updates to $OR(N)$.

B. Fulfillment of Obligations

The occurrence of actions that are required by concrete obligations in the system are monitored using the event $user\_Action$. This event is produced whenever some concrete obligation’s action is taken by the obligation’s subject on the obligation’s object. The event triggers an active rule which initiates the action $Fulfill$ to remove the concrete obligation from the state. This active rule is specified as follows:

\[
fulfill\_Obligation : Hold, (S, A, O, user\_Action) \\
initiates Fulfill(N, S, A, O) \\
\text{if } Obliged(N, S, A, O, Ctx, Ctx_v) \lor \text{OR}(N) \in \text{setGroupObligations}(N), \text{Obliged}(N, S, A, O, Ctx, Ctx_v) \in OR(N)
\]

Group obligation requirements are sets of alternative individual obligations. Therefore, the fulfillment of an obligation instance fulfills and removes all the group obligation requirements in which it appears. We formalize the effects of $Fulfill$ on the state in the following effect laws:

\[
Fulfill(N, S, A, O) \\
\text{causes } \neg\text{Obliged}(N, S, A, O, Ctx, Ctx_v) \\
\text{if } Obliged(N, S, A, O, Ctx, Ctx_v)
\]

\[
Fulfill(N, S, A, O) \\
\text{causes } \text{setGroupObligations}(N) = \text{setGroupObligations}(N) \setminus \text{OR}(N) \\
\text{if } OR(N) \in \text{setGroupObligations}(N), \text{Obliged}(N, S, A, O, Ctx, Ctx_v) \in OR(N)
\]

C. Violation of Obligations

The violation of a regular obligation instance is detected whenever its violation context is produced while the obligation holds as specified by the following active rule:

\[
violate\_Obligation : Hold, (S, A, O, start(Ctx_v)) \\
initiates Violate(N, S, A, O) \\
\text{if } Obliged(N, S, A, O, Ctx, Ctx_v)
\]

The action $Violate$ does not remove the obligation since it may still be required. However, violated obligations may be removed from the state by specifying that the violation of the obligation ends the obligation context.

The occurrence of the violation context of group obligation members does not necessarily violate any of the obligation requirements since a requirement is generally a set of alternative individual obligations. Therefore, the violation of a group obligation requirement is monitored as follows. Whenever the violation context of an obligation instance appearing in the requirement $OR(N)$ is detected, we add the fluent $Unfulfilled(N,S,A,O)$ to the state to denote that this particular individual obligation can no longer be fulfilled as follows:

\[
unfulfilled\_Obligation : Hold, (S, A, O, start(Ctx_v)) \\
initiates UnFulfill(N, S, A, O) \\
\text{if } OR(N) \in \text{setGroupObligations}(N), \text{Obliged}(N, S, A, O, Ctx, Ctx_v) \in OR(N) \\
\text{UnFulfill}(N, S, A, O) \\
\text{causes } \text{Unfulfilled}(N, S, A, O)
\]

The obligation requirement $OR(N)$ is violated whenever all individual obligations appearing in it become unfulfilled. This is specified using the following active rule.

\[
violate\_Requirement : Hold, (S, A, O, start(Ctx_v)) \\
initiates Violate(OR(N)) \\
\text{if } OR(N) \in \text{setGroupObligations}(N), \text{Obliged}(N, S, A, O, Ctx, Ctx_v) \in OR(N), \text{for all other Obliged}(N, S', A', O', Ctx, Ctx_v) \in OR(N) \text{Unfulfilled}(N, S', A', O')
\]

D. Deactivation of Obligations

Concrete obligations are deactivated when their context is ended while the obligation holds. The following active rule orders the action $UnOblige$ which removes the concrete obligation from the state when the obligation context ends:

\[
deactivate\_Obligation : Hold, (S, A, O, end(Ctx)) \\
initiates UnOblige(N, S, A, O) \\
\text{if } Obliged(N, S, A, O, Ctx, Ctx_v) \\
\text{UnOblige}(N, S, A, O) \\
\text{causes } \neg\text{Obliged}(N, S, A, O, Ctx, Ctx_v) \\
\text{if } Obliged(N, S, A, O, Ctx, Ctx_v)
\]

The deactivation of an obligation that is an instance of a group obligation removes the obligation instance from every requirement in which it appears as specified in the following effect law.

\[
UnOblige(N, S, A, O) \\
\text{causes } OR(N) = OR(N) \setminus \{\text{Obliged}(N, S, A, O, Ctx, Ctx_v)\} \\
\text{if } OR(N) \in \text{setGroupObligations}(N), \text{Obliged}(N, S, A, O, Ctx, Ctx_v) \in OR(N)
\]

After the deactivation of a group obligation instance, obligation requirements in which this instance appears are checked for violation. If all other individual obligations in the requirement are unfulfilled, the violation of the requirement is indicated by initiating the action $Violate$. To ensure that requirement violation is checked only after the deactivation of obligation instances, the active rule $check\_Req\_Violation$ is given lesser priority than the rule $deactivate\_Obligation$. The rule $check\_Req\_Violation$ is specified as follows:

\[
check\_Req\_Violation : Hold, (S, A, O, end(Ctx)) \\
\text{initiates Violate}(OR(N)) \\
\text{if } OR(N) \in \text{setGroupObligations}(N), \text{for all Obliged}(N, S, A, O, Ctx, Ctx_v) \in OR(N) \text{Unfulfilled}(N, S', A', O')
\]

E. Sanction Policies

Sanction policies are security rules which are activated after the detection of some policy rule violation. To activate sanction policies after the violation of a group obligation, we start violation contexts after the occurrence of the action $Violate(OR(N))$. For instance, a sanction policy which requires that every doctor who violates the group obligation $o_4$ should submit a report within 24 hours may be specified as follows:

\[
\text{Obligation} (\text{sanct}(o_4), \text{doctors}, \text{submit, report, violationSubject}(o_4), \text{Delay}(24\text{Hours}))
\]

Where the context $\text{violationSubject}$ is a context that is started after the violation of the obligation $o_4$. 

The sanction policy above applies to every subject who has contributed to the violation of the group obligation requirement $OR(N)$. Assume that the relation $\leq$ is a total order relation over elements in the set of doctors which reflects the hospital’s hierarchy, i.e. $d_1 \leq d_2$ if doctor $d_2$ is higher in the hospital’s hierarchy than doctor $d_1$. To specify that only the doctor who is highest in the hierarchy among the doctors who violated the requirement should submit the report, we may define the context $designatedViolationSubject$ as follows:

$$Hold_e(S, \_\_\_start(ViolationSubject(o_1)))$$

$$after \ Violate(OR(N))$$

$$if \ N = o_k, Obliged(N, S, A, O, Ctx, Ctx_o) \in OR(N)$$

$$⇒ Hold_e(S, \_\_\_start(designatedViolationSubject(o_k)))$$

$$after \ Violate(OR(N))$$

$$if \ N = o_k, Obliged(N, S, A, O, Ctx, Ctx_o) \in OR(N),$$

$$¬(Obliged(N, S', A', O', Ctx, Ctx_o) \in OR(N), S' \leq S')$$

Violation contexts, for instance $ViolationSubject$, may be ended after the fulfillment of the sanctioning obligation as follows:

$$Hold_e(S, \_\_\_end(violationSubject(o_k)))$$

$$after \ Fulfill(sanc(t(o_k)), S, submitt, report)$$

Our sanction policies may be activated for subjects other than those who contributed to the violation of the obligation. For instance, it is possible to specify sanction policies which apply for both the violation subject and his or her direct manager or only to the violation subject manager.

F. Constrained Existential Quantifiers

We may need to specify obligations which require a certain number of fulfillments such as “At least three doctors should reexamine a patient before pulling the plug on the patient’s life support”. The algorithm in figure 2 is extended in figure 3 to support the constrained existential quantifier where a number $m$ of required individual fulfillments can be specified. The extended algorithm handles the constrained existential quantifier by considering that the newly formed sentences $s_{n-1}(o, e_1, ..., e_{i-1}, E, e_{i+1}, ..., e_n)$ when the considered quantifier is $\exists m$ are produced in two steps. First, all combinations of $m$ sentences which can be made using the sentences $s_n(o, e_1, ..., e_{i-1}, E, e_{i+1}, ..., e_n)$ are generated, and elements of each combination are joined by $\land$. Then, these newly formed sentences are joined using the operator $\lor$. It may occur that the required number of individual fulfillments $m$ is impossible to reach. For instance, it may be required that three doctors check the patient whereas there are only two doctors at the hospital. In this case, the algorithm halts and an error is produced indicating that the group obligation cannot be fulfilled.

VI. Application Example

In this section, we reconsider the obligation presented informally in Section II. This obligation is specified as follows:

$$Obligation(o_4, doctors, reevaluate, patients, workingHours & assignedDoctor, end(doctorClinicHours) \lor$$

$$end(workingHours), groupContext(\lor/3/3a))$$

... while $N \geq 0$

$$for all Values of the parameter $s_n(e_i)$ do$$

$$if \ Q_n == exists(m) then$$

$$if \ m \leq \ number of instances of the form$$

$$s_n(e_1, ..., e_{i-1}, E, e_{i+1}, ..., e_n) then$$

$$for all possible combinations of $m$ sentences formed by$$

$$s_n(o, e_1, ..., e_{i-1}, E, e_{i+1}, ..., e_n) and joined by $\land$ do$$

$$s_{n-1}(o, e_1, ..., e_{i-1}, e_{i+1}, ..., e_n) \equiv \text{Join all combinations by $\lor$.}$$

else ERROR: obligation cannot be fulfilled

... Fig. 3. Support of the Constrained Existential Quantifier

The context $assignedDoctor$ is true whenever the doctor in the security rule is the doctor assigned to the patient in the security rule. The context $doctorClinicHours$ is a personalized context that represents the clinic hours of every doctor and is specified as follows for $d_1$:

$$Hold_e(d_1, \_\_\_start(doctorClinicHours))$$

$$after \ Clock \ if \ Time(Clock, 08 : 00)$$

$$Hold_e(d_1, \_\_\_end(doctorClinicHours))$$

$$after \ Clock \ if \ Time(Clock, 15 : 00)$$

The context $doctorClinicHours$ is similarly specified for the doctors ($d_2$, $d_3$) whose clinic hours are [08:00, 16:00] and [12:00, 19:00] respectively. We consider the following scenario. At 08:00, the context $workingHours$ becomes true. Assume that at this moment, patient $\{p_1\}$ is assigned to doctors $\{d_1, d_2\}$, patient $\{p_2\}$ is assigned to doctors $\{d_2, d_3\}$ and patient $\{p_3\}$ is assigned to doctor $\{d_3\}$. The activation of this group obligation creates the following group requirements:

$$OR_1 = (obl_1(d_1, readmit, p_1) \lor obl_2(d_1, discharge, p_1) \lor obl_3(d_2, readmit, p_1) \lor obl_4(d_2, discharge, p_1))$$

$$OR_2 = (obl_5(d_2, readmit, p_2) \lor obl_6(d_2, discharge, p_2) \lor obl_7(d_3, readmit, p_2) \lor obl_8(d_3, discharge, p_2))$$

$$OR_3 = (obl_9(d_3, readmit, p_3) \lor obl_10(d_3, discharge, p_3))$$

Once the group obligation is activated, subjects appearing in obligation instances of every group requirement are identified and notified of their obligations as follows: doctor $(d_1)$ is notified of the group obligation $OR_1$, doctor $(d_2)$ is notified of the group obligations $OR_1$ and $OR_2$, and doctor $(d_3)$ is notified of the group obligations $OR_2$ and $OR_3$.

At 11:00, doctor $d_2$ is assigned to the patient $p_3$. At this moment, $OR_3$ is updated and $(d_2, d_3)$ are notified of the new requirement $OR_3$. At 12:00, doctor $d_2$ discharges patient $p_3$ from the hospital. Consequently, $OR_3$ is fulfilled. At 13:00, doctor $d_2$ is de-assigned from the patient $p_1$. Consequently, obligations for doctor $d_2$ are removed from $OR_1$ and doctor $d_1$ is notified of the new requirement $OR_1$. At 14:00, doctor $d_1$ re-admits the patient $p_1$ to the hospital. Hence, $OR_1$ is fulfilled. At 16:00, the clinic hours of doctor $d_2$ ends. Thus, obligations for $d_2$, namely $obl_5$ and $obl_6$, are unfulfilled and...
doctor $d_3$ is notified that he or she is now the only subject who may fulfill $OR_2$. Thus, if doctor $d_3$ reevaluates patient $p_2$ before the end of the context $workingHours$, $OR_2$ is fulfilled; otherwise, $OR_2$ is violated and doctors $(d_2, d_3)$ are considered co-responsible of its violation. Subsequently, sanctions, in the form of new obligations or prohibitions, may be activated for doctors $(d_2,d_3)$.

VII. RELATED WORK

In access control policies, group relations such as roles [1] and domains [2] have been essentially used to simplify policy management. Several works have studied usages of roles in access control policies. For instance in [12], the specification of separation of duty constraints and hierarchies using roles is studied. In this paper, we study group relations in obligation policies and show how they can enhance the expressiveness of obligations by enabling the specification of shared responsibilities and alternative duties.

Several policy languages support the specification of obligations. PONDER [6] and PDL [7] support the specification of obligations in the form of Event-Condition-Action (ECA) rules. These obligations are used to trigger the immediate execution of actions after the detection of some events. Therefore, these languages do not consider the monitoring nor the violation of obligations. Obligations in the form of ECA rules are thus more suitable for the specification of system management operations. In this paper, we focus on user-obligations and we use ECA rules to formalize the management and monitoring of obligations. In PONDER, security rules may be specified on the group level using domains. Conflict detection and resolution for domain-based policies is studied in [13].

Rei [8] supports the specification of user-obligations. In Rei, obligations are not associated with a deadline. Therefore, the policy engine can only infer whether obligations have been fulfilled or not. Rei does not support the expression of policies on the group level. However, role-based policies may be specified using conditions over subject attributes.

LGI [14] is a policy language for distributed environments. Policy rules in LGI are specified directly in prolog and the language does not support group relations. The support of role-based policies in LGI is discussed in [15]. UCON [4] presents a family of models to express pre and ongoing usage controls. In UCON, obligations correspond to actions that users are required to take before or while having an access to system resources.

We conclude that, to our knowledge, this is the first time the notion of group obligations is introduced and formalized.

VIII. CONCLUSION

In this paper, we have introduced the notion of group obligations and showed how they enhance the expressiveness of obligation languages by enabling the specification of shared responsibilities and alternative duties. We have discussed group obligation management requirements using a reasonably rich contextual obligation model where obligations can be violated, fulfilled and deactivated. We have also briefly discussed the activation of sanction policies after the violation of group obligations.

In our work, we have made the assumption that group obligations are going to be most likely specified for small groups of subjects, actions and objects. This assumption is justified by the fact that a large group of subjects will not be able to coordinate the fulfillment of their shared obligations. Therefore, future work consists of studying possible optimizations to the presented algorithms. Another possible direction is to study the specification of group obligations that apply to non atomic actions [16]. We also intend to study the modeling of consent. Consent is one of the requirements that is relevant to the preservation of subjects privacy. Consent may be represented using an obligation whose subject is allowed to take several actions ($\text{deny, accept, accept\_Once, ...}$).

ACKNOWLEDGMENT

This work is partially funded by the FLUOR project (ANR-07-SESU-005-01). Yehia El Rakaiby’s PhD thesis is funded by a grant from the Brittany region.

REFERENCES