Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 186 (2007) 3–26

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

High Level Conflict Management Strategies in Advanced Access Control Models

Frédéric Cuppens[a] Nora Cuppens-Boulahia[a]
Meriam Ben Ghorbel[a,b]

[a] GET/ENST Bretagne, 2 rue de la Châtaigneraie, 35512 Cesson Sévigné Cedex, France

[b] SUPCOM Tunis, Route de Raoued Km 3.5, 2083 Ariana, Tunisie

Abstract

Specifying a security policy that includes both permissions and prohibitions, may lead to conflicts. This corresponds to a situation where a subject is both permitted and prohibited to perform a given action on a given object. We adopt a comparative approach to investigate this problem. We first investigate access control models based on rules, called Rule-BAC, and present weaknesses that arise when we try to manage conflicts in this model. In particular, Rule-BAC models fail to provide decidable solution to redundant rules and potential conflicts problems. Then, we show how a more structured model, say OR-BAC (Organization Based Access Control), gifted with inheritance mechanism make redundant rules and potential conflict problems tractable in polynomial time.

Keywords: Rule-BAC, Or-BAC, Potential Conflict, Prioritized policy.

1 Introduction

Access control is modelled as a set of authorizations specified either by a security officer or a private user in accordance with some security policy. Usually, these authorizations specify that a given subject (a user or a process) is permitted to perform a given action (an access mode) on a given object (a resource of the system). This static authorization triple $\langle subject, action, object \rangle$ is suited for traditional environments and applications but is less appropriate to meet requirements of the rising systems. Indeed, there is a need of more expressiveness, that is other kinds of authorizations must be supported: content based authorization, constraint based authorization and more particularly negative authorization. In that way the security officer is given means to specify general contextual permission rules and associate exceptions to these general rules using prohibitions. For instance, a nurse may be permitted to consult a medical record (general rule) except the physician's private comments (exception corresponding to prohibition). Moreover, in an access control

doi:10.1016/j.entcs.2007.01.064

model where hierarchies and inheritance mechanisms are included, prohibitions can be used to regulate the inheritance policy of permissions.

However, when an access control model includes the possibility to specify both permissions and prohibitions, some conflicts may occur. This is the case when a subject is both permitted and prohibited to perform an action on an object. Hence, the system might not be able to decide either to allow or deny the access. This problem was investigated by several models (for instance [2,9,7,3,16,5,15,1]). The conflict issue must be addressed by defining a conflict resolution strategy. A conflict resolution strategy consists in a set of rules that enable the system to decide, in case of a conflict, to discard either the positive or the negative authorization. Therefore, the resulting access control policy will depend on the chosen conflict resolution policy. Thus the security officer should have the possibility to define his or her own conflict resolution strategy in order to obtain a relevant access control policy.

Rule based access control (Rule-BAC) termed models [7,3,15,4,13] take the lead of access control models attempting to meet expressiveness requirements and offer means to solve conflict problems. In this kind of model, access control is defined as a set of rules $Condition \rightarrow Authorization$ where $Condition$ is a set of constraints over the subjects, actions and objects. In this paper, we begin with analyzing conflict management in the context of Rule-BAC Model and show that there are several problems this model fails to solve. First, assigning higher priorities to some access rules to manage conflicts can lead to the emergence of rules that never apply, say redundant rules. Unfortunately, checking the non-redundancy condition is undecidable and there is a lack of replacement solution in the Rule-BAC model. The second unsolved problem is that current solutions only manage *actual* conflicts but they are unable to detect *potential* conflicts, that is the coexistence of rules that lead to some conflicts if their associated conditions are simultaneously satisfied. Managing potential conflicts is important since we gain the guarantee that actual conflict will never occur. However, checking the potential conflict condition in the Rule-BAC model is also undecidable. We argue that the main reason of these drawbacks is a structure lack of Rule-BAC models.

We then analyze these problems in the context of the Or-BAC model [17] and show how all of them are formally and effectively solved. In Or-BAC, which is a structured and more expressive model built on top of Rule-BAC, specifying an access control policy is centered around the concept of *organization*. Each organization can specify its own security policy at an "organizational" level. For this purpose, a policy specifies that some *roles* are permitted or prohibited to perform some *activities* on some *views*. These concepts of role, activity and view are used to specify the policy independently from concrete implementation of subjects, actions and objects in the system. Moreover, Or-BAC model gives means to specify contextual authorizations.

The approach used to manage conflicts in Or-BAC is based on assigning priorities to access control rules as we suggest in Rule-BAC. Nevertheless, to overcome difficulties encountered in Rule-BAC, we restate the problems of rule redundancy and potential conflicts using inheritance mechanisms and separated constraints specification [14]. We then show that, using this approach, rule redundancy and potential conflicts are tractable problems computable in polynomial time.

The remainder of this paper is organized as follows. The basic Rule-BAC model and associated conflict management are introduced in section 2. Then, in Section 3, we define the prioritized Rule-BAC model and show how more expressive conflict resolution strategies may be defined in this model. In this section, we also bring out redundancy and potential conflict problems that this model fails to solve. Section 4 is dedicated to the Or-BAC model. We introduce in this section the model and its conflict management principles. We emphasize on the hierarchy structure of this model. Then we show in section 4.3 and section 4.4 how the Or-BAC components and its hierarchy structure are used in the authorization derivation mechanism and separated constraint specification to succeed in solving the undecidability problems of redundant rules and potential conflict detection. Section 5 is a discussion about related works and section 6 concludes the paper.

2 Rule Based Access Control

2.1 Rule-BAC principles

In Rule-BAC models, a policy is seen as a set of access control rules in which some conditions drive a positive or negative authorizations:

$condition \rightarrow Permission(Subject, Action, Object)$ or,

$condition \rightarrow Prohibition(Subject, Action, Object)$

In the Rule-BAC model, a conflict occurs when it is possible to apply two different access control rules to derive that a given subject is both permitted and prohibited to perform a same action on a same object:

$\exists s, \exists a, \exists o, Permission(s, a, o) \wedge Prohibition(s, a, o)$

When implementing a given policy, conflicts must be solved, else it should not be possible to decide if a given access must be granted or denied when a conflict occurs. The approach to solve conflicts consists in defining a conflict resolution strategy (CRS). The basic principle of a CRS is to consider that predicates *Permission* and *Prohibition* only define *prima facie* authorizations. The objective of a CRS is to derive *actual* authorizations from *prima facie* authorizations. For this purpose, we shall consider predicates *A-Permission* and *A-Prohibition* to represent actual positive or negative authorizations. Thus, a CRS will be defined as a set of rules having the predicates *A-Permission* or *A-Prohibition* in the conclusion.

Let us now present a formalization of Rule-BAC principles.

2.2 Rule-BAC model

This section is dedicated to the formal definition of the Rule-BAC model. In the Rule-BAC model, a policy is specified in first order logic language. This language should include the following predicates.

Definition 1: We introduce four kinds of predicates.

- *Domain predicates.* Predicates used to model the IT system to which the policy applies. Thus, domain predicates depend on the application field.

- *Prima Facie access control predicates.* Predicates used to express prima facie positive and negative authorizations: $Permission(x, y, z)$ and $Prohibition(x, y, z)$.
- *Actual access control predicates.* Predicates used to express actual positive and negative authorizations: $A\text{-}Permission(x, y, z)$ and $A\text{-}Prohibition(x, y, z)$.
- *Constraint violation predicate.* Predicate used to express a constraint violation: $error()$.

Definition 2: The domain to which the policy applies is represented by a finite set of domain formulae Dom, $Dom = Rdom \cup IDom$, where:

- $IDom$ (domain instance) is a finite set of facts $P(t_1, ..., t_n)$ where P is a domain predicate.
- $RDom$ (domain rules) is a finite set of closed domain formulae, that is formulae that only contain domain predicates. For example: $\forall x, surgeon(x) \rightarrow physician(x)$

Definition 3: The domain constraints Cte is a finite set of formulae having the form $condition \rightarrow error()$ where $condition$ represents a generic well-formed formula.
Definition 4: An *access control policy* ACP is a finite set of well-formed closed formulae such as:

- $\forall s, \forall a, \forall o, Condition \rightarrow Permission(s, a, o)$ or
- $\forall s, \forall a, \forall o, Condition \rightarrow Prohibition(s, a, o)$

where $Condition$ is a conjunctive formula $P_1 \wedge ... \wedge P_n$ and each P_i represents a generic well-formed formula.
Definition 5: A *conflict resolution strategy* CRS is a finite set of rules having the predicates $A\text{-}Permission$ or $A\text{-}Prohibition$ in the conclusion.
Definition 6: A *regulated system* Sys is a logical theory defined as follows:

- $Sys = ACP \cup CRS \cup Dom \cup Cte$

Assumption 1: In the following, we suppose that there is no violated constraint, that is, the constraint predicate $error()$ can never be derived from the regulated system Sys: $Sys \nvdash error()$.

The main drawback of full first order logic is that it leads to undecidable theory. Thus, we have to restrict the regulated system definition to obtain a decidable and tractable theory. One way to proceed is to consider that a regulated system corresponds to a stratified Datalog program [18]. Stratifying a Datalog program consists in ordering rules so that if a rule contains a negative literal then the rule that defines this literal is computed first. A stratified Datalog program is computable in polynomial time.

In the following, we assume that a Rule-BAC policy corresponds to a stratified Datalog program and similarly for the Prioritized Rule-BAC and Or-BAC models presented in sections 3 and 4.

2.3 Conflict management

In a Rule-BAC policy, a *prima facie* conflict appears when prima facie permission and prohibition are assigned to the same subject, action and object.

Definition 7: The access control policy ACP of a regulated system Sys is a prima facie conflicting policy iff: $Sys \vdash \exists s, \exists a, \exists o, Permission(s,a,o) \wedge Prohibition(s,a,o)$.

As we have given the definition of a conflict resolution strategy, we can now give a definition of an *actual* conflict.

Definition 8: The access control policy ACP of a regulated system Sys managed by a conflict resolution strategy CRS is an actual conflicting policy iff:

- $Sys \vdash \exists s, \exists a, \exists o, A\text{-}Permission(s,a,o) \wedge A\text{-}Prohibition(s,a,o)$

Theorem 1: If Sys is a regulated system that corresponds to a Datalog program, then we can decide that Sys is an actual conflicting policy in polynomial time.

Proof: The proof is trivial. Regulated systems correspond to stratified Datalog programs, which are computable in polynomial time. □

The main idea is that a conflict resolution strategy does not necessarily guarantee that all conflicts will be solved. Thus, there might remain some conflicts between some actual authorizations. If it can be proved that a strategy solves all conflicts then it is called an *effective strategy.*

Definition 9: A conflict resolution strategy is effective if it guarantees that any regulated system to which it is applied does not contain any actual conflict.

Theorem 2: If CRS is a conflict resolution strategy, then checking that CRS is an effective strategy is generally undecidable.

Proof: Let CRS be some conflict resolution strategy. Then to prove that this strategy is effective, we have to prove that, for every system Sys_{CRS} regulated by CRS:

$Sys_{CRS} \vdash \neg(\exists s, \exists a, \exists o, A\text{-}Permission(s,a,o) \wedge A\text{-}Prohibition(s,a,o))$

However, we know that proving that a given formula is a theorem of first order logic is generally undecidable. □

2.4 Example

We consider a example of simple conflict management strategy $S1$:

- $S1$: Prohibitions take precedence
  $Prohibition(s,a,o) \rightarrow A\text{-}Prohibition(s,a,o)$
  $Permission(s,a,o) \wedge \neg Prohibition(s,a,o) \rightarrow A\text{-}Permission(s,a,o)$

If $S1$ is applied to a stratified policy, this policy remains stratified. Actually in $S1$, *Prohibition* is in a lower stratum than *Permission*.

Let us now consider an example of regulated system:

R1: $nurse(x) \wedge medical\_record(y) \rightarrow Prohibition(x, read, y)$
"A nurse is prohibited to read any medical record"

R2: $nurse(x) \wedge medical\_record(y) \wedge patient\_record(y,z) \wedge urgent\_case(z)$
$\rightarrow Permission(x, read, y)$

"In case of emergency, a nurse is permitted to read the patient's medical record"

R3: $physician(x) \wedge medical\_record(y) \wedge$
$patient\_record(y,z) \wedge attending\_physician(z,x)$
$\rightarrow Permission(x, read, y)$

"A physician is permitted to read the medical records of her own patients"

R4: $physician(x) \wedge medical\_record(y) \wedge suspended(x)$
$\rightarrow Prohibition(x, read, y)$

"A physician is prohibited to read any medical record as long as she is suspended"

IDOM: $physician(John) \wedge suspended(John) \wedge$
$nurse(Peter) \wedge medical\_record(doc\_31) \wedge patient\_record(doc\_31, Mary) \wedge$
$attending\_physician(Mary, John) \wedge urgent\_case(Mary)$

The following prima facie authorizations are derived from this policy:

- $Prohibition(Peter, read, doc\_31)$ (from R1)
- $Permission(Peter, read, doc\_31)$ (from R2)
- $Permission(John, read, doc\_31)$ (from R3)
- $Prohibition(John, read, doc\_31)$ (from R4)

The first authorization is conflicting with the second one and it is also the case for the third and the fourth authorizations. Let us apply strategy $S1$ to this policy. We obtain the following actual authorizations:

- $A\text{-}Prohibition(Peter, read, doc\_31)$
- $A\text{-}Prohibition(John, read, doc\_31)$

Using strategy $S1$, we notice that rule $R1$ always takes precedence over rule $R2$. Thus, $R2$ is useless as it is never applied. This shows that strategy $S1$ is too coarse.

In the following section, we consider conflict management strategies that provide means to define priorities over the authorizations in a much more elaborate way.

3 Prioritized Rule-BAC

3.1 Principle

We associate the authorization rules with priorities in order to evaluate their significance in conflicting situations. Priorities between access control rules may be sometimes derived from the rules syntactical format. For instance, let us come back to rules $R1$ and $R2$ of our previous example. If we consider that rule $R1$ has higher priority than rule $R2$, then $R2$ will never apply and is therefore useless. In this case, we shall say that rule $R2$ is a *redundant rule.* To avoid this misspecification, we can conclude that rule $R2$ must be associated with higher priority than rule $R1$. In the following section, we shall formally define this notion of redundant rule.

However, automatic management of priorities based on rule syntactical format is not always possible and it is sometimes necessary to explicitly assign priorities to rules to solve conflicts. For instance, consider the following access control rule:

R5: $nurse(x) \wedge medical\_record(y) \wedge suspended(x) \rightarrow Prohibition(x, read, y)$
"A suspended nurse is prohibited to consult any medical record"

Let us now assume that Peter is a suspended nurse and that Mary is an urgent case. Which rule applies to this situation? Is it rule $R2$? In this case, we shall conclude that Peter is permitted to read Mary's medical record. Or rule $R5$ so that we shall conclude that Peter is prohibited to read Mary's medical record? In this example, we cannot use the rule syntactical format to decide which rule applies. Explicit priorities must be assigned to rules $R2$ and $R5$ to solve the conflict.

Notice also that if there is no suspended nurse or no urgent patient case, then it is no longer possible to apply both rules $R2$ and $R5$. So, in this situation, there is no *actual* conflict. However, in this situation, we shall say that there is a *potential* conflict between rules $R2$ and $R5$, that is a conflict will occur when a suspended nurse and an urgent patient case are both inserted in the domain instance $IDom$. In the following, we shall formally define this notion of potential conflict.

3.2 Prioritized Rule-BAC model

We consider Π a finite set of priorities. We assume that Π is associated with a partial order relation $\prec$. If p_1 and p_2 are two priorities, then $p_1 \prec p_2$ means that p_2 has higher priority than p_1.

The Prioritized Rule-BAC model is then defined as follows:

Definition 10: This definition is similar to definition 1 except that we replace:
3-place predicate $Permission(subject, action, object)$ by
4-place predicate $Permission(subject, action, object, priority)$, and
3-place predicate $Prohibition(subject, action, object)$ by
4-place predicate $Prohibition(subject, action, object, priority)$.

We also introduce two new predicates *Higher-Permission* and *Higher-Prohibition*. $Higher\text{-}Permission(s, a, o, p)$ means that there exists a permission for subject s, action a and object o having a priority higher than p. $Higher\text{-}Prohibition(s, a, o, p)$ means that there exists prohibition over s, a and o having a priority higher than p.

Definition 2 and 3 remains unchanged except that $RDom$ and Cte include the following rules:

- $Permission(s, a, o, p_2) \wedge p_2 \prec p_1 \rightarrow Higher\text{-}Permission(s, a, o, p_1)$
- $Prohibition(s, a, o, p_2) \wedge p_2 \prec p_1 \rightarrow Higher\text{-}Prohibition(s, a, o, p_1)$
- $(p_1 \prec p_2 \wedge p_2 \prec p_3) \rightarrow p_1 \prec p_3$ (transitivity of $\prec$)
- $(p_1 \prec p_2 \wedge p_2 \prec p_1) \rightarrow error()$ (antisymmetry of $\prec$)

Definition 4, 5 and 6 are respectively replaced by definition 11, 12 and 13:

Definition 11: A *prioritized access control policy PACP* is a finite set of well-formed closed formulae defined as follows:

- $\forall s, \forall a, \forall o, \forall p, Condition \rightarrow Permission(s,a,o,p)$ or
- $\forall s, \forall a, \forall o, \forall p, Condition \rightarrow Prohibition(s,a,o,p)$

Definition 12: A *global conflict resolution strategy GCRS* corresponds to the following two rules:

- $Permission(s,a,o,p) \wedge \neg Higher\text{-}Prohibition(s,a,o,p)$
  $\rightarrow A\text{-}Permission(s,a,o)$
- $Prohibition(s,a,o,p) \wedge \neg Higher\text{-}Permission(s,a,o,p)$
  $\rightarrow A\text{-}Prohibition(s,a,o)$

The first rule states that an actual permission can be derived from a prima facie permission if there is no higher prima facie prohibition in the access control policy. The second rule is similar but for deriving actual prohibitions.

Definition 13: A *regulated system Sys* is a logical theory defined as follows:

- $Sys = PACP \cup GCRS \cup Dom \cup Cte$

3.3 Conflict management

In Prioritized Rule-BAC, definition of a *prima facie* conflict is similar to definition 7 except that we have to insert priorities. This leads to definition 14.

Definition 14: The prioritized access control policy $PACP$ of a regulated system Sys is a prima facie conflicting policy iff:
$Sys \vdash \exists s, \exists a, \exists o, \exists p_1, \exists p_2, Permission(s,a,o,p_1) \wedge Prohibition(s,a,o,p_2)$.

We also specify as a constraint, called CP (for Consistent Priority), that a prima facie conflict cannot occur with the same priority level:

CP: $Permission(s,a,o,p) \wedge Prohibition(s,a,o,p) \rightarrow error()$

If CP is violated, it is no longer possible to apply ordered priorities to solve the conflict since the conflicting permission and prohibition have equal priorities.

Definition of actual conflict in Prioritized Rule-BAC is similar to definition 8. We can then prove the following theorem:

Theorem 3: If the constraint CP is never violated and if the set Π of priority levels is associated with a *total* order relation, then the global conflict resolution strategy $GCRS$ is effective.

Proof: Let us assume that the global conflict resolution strategy GCRS is not effective. It follows from definition 9 that there is an actual conflict such that $\exists s, \exists a, \exists o, A\text{-}Permission(s,a,o) \wedge A\text{-}Prohibition(s,a,o)$. The only way to derive these facts is by applying the following rules (definition 12) :

$$Permission(s,a,o,p) \wedge \neg Higher\text{-}Prohibition(s,a,o,p) \rightarrow A\text{-}Permission(s,a,o) \quad (1)$$

$$Prohibition(s,a,o,p) \wedge \neg Higher\text{-}Permission(s,a,o,p) \rightarrow A\text{-}Prohibition(s,a,o) \quad (2)$$

Rule 1 means that there is a priority level p_1 such that $Permission(s,a,o,p_1)$ and there is no p'_1 such that $(p_1 \prec p'_1 \wedge Prohibition(s,a,o,p'_1))$. Similarly rule 2 means that there is a priority level p_2 such that $Prohibition(s,a,o,p_2)$ and there is no p'_2 such that $(p_2 \prec p'_2 \wedge Permission(s, a, o, p'_2))$.

So, since the constraint CP cannot be violated, we cannot have $p_1 = p_2$. So, we can conclude that we have $(Permission(s, a, o, p_1) \wedge Prohibition(s, a, o, p_2) \wedge \neg(p_1 \prec p_2) \wedge \neg(p_2 \prec p_1))$. But, this is in contradiction with the assumption that the set Π of priority levels is associated with a total order relation. □

Notice that this theorem does not apply if the set of priority levels is associated with a *partial* order relation. Unfortunately, a partial order relation is more flexible than a total order because it is not necessary to compare the priority of every pair of access control rules. For instance, let us consider rules $R1$ and $R3$. Rule $R1$ applies to nurses whereas rule $R3$ applies to physicians. Let us assume that there is a constraint that specifies that a subject cannot be both a nurse and a physician:

C1: $\forall x, (nurse(x) \wedge physician(x)) \rightarrow error()$

In this case, it is never possible to apply both rules $R1$ and $R3$ to derive a conflict. We shall say that $R1$ and $R3$ are *unrelated*. It is not necessary to compare the priority of unrelated rules and therefore a partial order relation on priorities is sufficient in this case. Unrelated rules are further formalized in section 3.5.

3.4 Redundant rules

Let us consider rules $R1$ and $R2$. We can consider that $R1$ specifies a general case (Nurses are prohibited to read medical records) whereas rule $R2$ specifies an exception to rule $R1$ (Nurses are actually permitted to read medical record *in case of emergency*). Exceptions are formalized as follows. Let us consider two rules R_i and R_j and let $condition(R_i)$ and $condition(R_j)$ be the conditions of R_i and R_j.

Definition 15: Rule R_i is an exception to rule R_j if we can prove that:

$(RDom \cup Cte^*) \vdash condition(R_i) \rightarrow condition(R_j)$

where Cte^* is defined as follows:

$\neg condition \in Cte^*$ iff $(condition \rightarrow error()) \in Cte$

For instance, if we assume that the above constraint $C1$ belongs to Cte then formula $\neg(nurse(x) \wedge physician(x))$ belongs to Cte^*.

Definition 16: Rule R_i is a *strict* exception to rule R_j if R_i is an exception to rule R_j and R_j is not an exception to rule R_i.

Now, if rule R_i is a strict exception to rule R_j, then R_i should be assigned higher priority than rule R_j, else R_i never applies and is therefore a redundant rule. We can give the condition to prevent redundant rules. Let us consider two rules R_i and R_j and let $Priority(R_i)$ and $Priority(R_j)$ be the priorities of R_i and R_j.

Non-redundancy condition: If rule R_i is a strict exception to rule R_j then $Priority(R_j) \prec Priority(R_i)$.

Recognizing redundant rules is clearly an important problem. Unfortunately, we can prove the following theorem:

Theorem 4: Checking that rule R_i is an exception to rule R_j is generally undecidable.

Proof: As for theorem 2, the proof of theorem 4 is based on the undecidability of first order logic.

So, let $condition(R_i)$ and $condition(R_j)$ be respectively the conditions of rules R_i and R_j. We have to show that proving:

$(RDom \cup Cte^*) \vdash condition(R_i) \rightarrow condition(R_j)$

is generally undecidable.

For this purpose, we show that $(RDom \cup Cte^*)$ actually corresponds to any theory in first order logic.

So let F be a first order formula built using some domain predicates of the regulated system.

If F respects the Datalog restriction, F will be included in $RDom$.

If F does not respect the Datalog restriction, we can write a constraint of Cte having the form $\neg F \rightarrow error()$. For instance, let us consider the formula $\forall x, Nurse(x) \rightarrow \neg Physician(x)$ (a nurse is not a physician). Due to the negation in the conclusion, this is not a Datalog formula. However, we can include in Cte the following constraint:

$\forall x, (Nurse(x) \wedge Physician(x)) \rightarrow error()$

However, when building Cte^*, the formula F will be inserted in the theory $(RDom \cup Cte^*)$. So, $(RDom \cup Cte^*)$ can actually include any first order formula, and thus proof in this theory is generally undecidable.

As a consequence, checking the non-redundancy condition is also undecidable. □

3.5 *Potential conflict detection*

Definition 8 provides means to derive actual conflicts. However, in Rule-BAC the set of authorizations may change dynamically as the authorization conditions are updated. In the example of section 2.4, if physician *John* is no longer suspended, rule $R3$ does not apply to *John* anymore.

The administrator of an access control policy may want to establish the rule pairs that may lead to conflict situations. In our example, rules $R2$ and $R5$ do not generate a conflict as long as nurse Peter is not suspended. Such a situation is called a *potential conflict*. To recognize potential conflict situations, we need to model the notion of unrelated rules suggested in section 3.3.

Definition 17: Rules R_i and rule R_j are unrelated iff we can prove that:

$(RDom \cup Cte^*) \vdash \neg(condition(R_i) \wedge condition(R_j))$

We can now specify a condition for situations of potential conflict.

Potential conflict condition: (first definition) There is a potential conflict between two rules R_i and R_j if:

- R_i derives a permission and,
- R_j derives a prohibition and,
- R_i and R_j are *not* two unrelated rules and,
- $Priority(R_i)$ and $Priority(R_j)$ are not comparable.

We can prove the following theorem:

Theorem 5: Let us consider a given prioritized access control policy $PACP$ of a regulated system Sys. If there is an actual conflict in Sys, then there is a potential conflict between two access control rules of $PACP$.

Proof: Let us assume that there is an actual conflict and there is no potential conflict between two access control rules of PACP.

An actual conflict occurs when (definition 8):
$\exists s, \exists a, \exists o, A\text{-}Permission(s, a, o) \wedge A\text{-}Prohibition(s, a, o)$

It follows from definition 12 that an actual conflict occurs when:

$$Permission(s, a, o, p_1) \wedge \neg Higher\text{-}Prohibition(s, a, o, p_1) \wedge$$
$$Prohibition(s, a, o, p_2) \wedge \neg Higher\text{-}Permission(s, a, o, p_2) \quad (1)$$

We can thus conclude that there are two rules R_i and R_j such that R_i derives $Permission(s, a, o, p_1)$ and R_j derives $Prohibition(s, a, o, p_2)$ and that R_i and R_j are not unrelated rules. Now, assume that there is no potential conflict between R_i and R_j. It follows from the first definition of potential conflict condition the following rule:

$$Permission(s, a, o, p_1) \wedge Prohibition(s, a, o, p_2) \wedge$$
$$(Higher\text{-}Prohibition(s, a, o, p_1) \vee Higher\text{-}Permission(s, a, o, p_2)) \quad (2)$$

We can conclude that the assumption there is no potential conflict given by rule (2) is in contradiction with the assumption there is an actual conflict given by rule (1). □

We can actually find a weaker potential conflict condition than the one suggested above. To illustrate the problem, let us consider the following access control rule:

R6: $nurse(x) \wedge medical\_record(y) \wedge$
$patient\_record(y, z) \wedge urgent\_case(z) \wedge suspended(x)$
$\rightarrow Permission(x, read, y)$

In this case, rule R6 acts as an exception to rule R5 and thus it must have higher priority than rule R5 if the non redundant rule condition presented in section 3.4 is satisfied. Now, the potential conflict between rules $R2$ and $R5$ is solved: due to rule $R6$, we can conclude that a nurse is permitted to read the medical record of an urgent case patient, even if this nurse is suspended.

Therefore, we can relax the potential conflict condition as follows.

Potential conflict condition: (second definition) There is a potential conflict between two rules R_i and R_j if:

- R_i derives a permission and,
- R_j derives a prohibition and,
- R_i and R_j are *not* two unrelated rules and,
- there is no rule R_k such that:
  - · $(RDom \cup Cte^*) \vdash (condition(R_i) \wedge condition(R_j)) \rightarrow condition(R_k)$
  - · and ((R_k derives a prohibition and $Priority(R_i) \prec Priority(R_k)$) or ($R_k$ derives a permission and $Priority(R_j) \prec Priority(R_k)$))

We can still use this second condition to prove theorem 5.

Proof: First steps of the proof are similar to the first definition. If we assume that there is an actual conflict, then we can conclude that there are two rules R_i and R_j such that R_i derives $Permission(s, a, o, p_1)$ and R_j derives $Prohibition(s, a, o, p_2)$ and that R_i and R_j are not unrelated rules. Now, assume that there is no potential conflict between R_i and R_j according to the second definition. This means that there is a rule R_k such that $(condition(R_i) \wedge condition(R_j)) \rightarrow condition(R_k)$.

Thus we can apply rule R_k and derive (p_3 represents $priority(R_k)$):

$$(Prohibition(s, a, o, p_3) \wedge Higher\text{-}Prohibition(s, a, o, p_1)) \vee$$
$$(Permission(s, a, o, p_3) \wedge Higher\text{-}Permission(s, a, o, p_2)) \quad (3)$$

So, we can conclude that the assumption there is no potential conflict given by rule (3) is in contradiction with the assumption there is an actual conflict. □

Unfortunately, we can also state the following theorem:

Theorem 6: Checking that two rules R_i and R_j are unrelated is generally undecidable.

Proof: Proof is similar to theorem 4. □

As a consequence, checking the potential conflict conditions (first or second condition) is also undecidable.

3.6 Example

We consider the access control policy corresponding to rules $R1$ to $R5$ and we assume that Π is $\{p1, p2, p3\}$ with $p1 \prec p2 \prec p3$. The access control rules are associated with the following priorities: (1) $R1$ and $R3$ have priority $p1$, (2) $R2$ and $R4$ have priority $p2$ and (3) $R5$ has priority $p3$.

We also assume that constraint $C1$ is not violated so that a subject cannot be both a nurse and a physician.

One can check that $R2$ is a strict exception to $R1$. Applying the non redundancy condition, we can conclude that $R2$ must have higher priority than $R1$. This is compatible with the assumption that $p1 \prec p2$. Similarly, $R5$ is a strict exception to $R1$ which is compatible with the assumption that $p1 \prec p3$.

$R3$ and $R4$ are not unrelated rules. However, since $R4$ has higher priority than rule $R3$, potential conflict is solved. A similar comment applies to rules $R2$ and $R5$.

Finally, due to constraint $C1$, rules $R1$ and $R3$ are unrelated. So, there is no potential conflict between these two rules. Similarly, $R2$ and $R4$ are unrelated rules and $R3$ and $R5$ are unrelated rules.

We can thus conclude that there is no redundant rule and no potential conflict in this prioritized access control policy. Thus, using the contraposition of theorem 5, we can derive that there will be no actual conflict in every system regulated by this access control policy.

Another way to derive the same result is by using theorem 3. Since Π is associated with a total order relation, we have simply to check that constraint CP cannot be violated. The only way to violate CP is by applying both rules having the same priorities, that is rules $R1$-$R3$ or rules $R2$-$R4$. Since rules $R1$-$R3$ and rules $R2$-$R4$

are both unrelated, this is not possible. As a consequence, we can conclude that this strategy is effective and no actual conflict is possible.

3.7 Summary

Using the Prioritized Rule-BAC model, we have presented two different ways to prevent actual conflicts.

The first approach is by showing that the strategy is effective. However, this approach is restricted to access control policies that are associated with a total order relation. Moreover, we have to check that constraint CP is not violated.

The second approach is by checking the Potential Conflict Condition. This approach may be used when a partial order relation is used. Unfortunately, checking the Potential Conflict Condition is generally undecidable.

In both approaches, we also show how to detect redundant rules. This is important since redundant rules often correspond to specification errors. Indeed, it is not "normal" that an administrator includes useless rules in the access control policy. However, checking the Non-Redundancy Condition is also generally undecidable.

Furthermore, there is no simple way in Prioritized Rule-BAC to specify automatic priority assignment strategies. For instance, it may be useful to specify that access control rules related to emergency situations have higher priority than other access control rules.

In the following section, we present the Or-BAC model and show how this model provides solutions to these different issues, namely we restate the potential conflict and redundant rules problems so that that these problems are decidable and tractable in polynomial time. We also show how to specify automatic priority assignment strategies in the Or-BAC model.

4 Conflict management in the Or-BAC model

4.1 Basic Principles of Or-BAC

In Or-BAC, the expression of an access control policy is further structured than in the Rule-BAC model. This will provide means to restate the problems of redundant rules and potential conflicts in a decidable and tractable way. More precisely, Rule-BAC policies consist of rules in which a conjunction of conditions leads to a permission or prohibition. After analyzing the structure of the *condition* of a Rule-BAC policy rule, we suggest structuring this *condition* as follows:

- $condition \leftrightarrow$
  $cond\_subject(s) \wedge cond\_action(a) \wedge cond\_object(o) \wedge\ constraint(s, a, o)$

where $cond\_subject(s)$, $cond\_action(a)$ and $cond\_object(o)$ are respectively the conditions the subject s, the action a and the object o must separately satisfy so that the corresponding rule applies. $constraint(s, a, o)$ is an additional condition that joins subject s, action a and object o. Satisfying the constraint is necessary to activate the rule.

For instance, let us consider the rule "a physician is permitted to consult his or her patient's medical record". In this case, $cond\_subject(s)$, $cond\_action(a)$ and $cond\_object(o)$ respectively correspond to the conditions that s is a physician, a is an action of consulting and o is a medical record. $constraint(s, a, o)$ is a condition that joins subject s and object o - in this example, action a is absent from the constraint - namely o must be a record of s's patient. The main idea is to define a language that enables to structure the set of conditions on which an authorization is granted. This is one of the main purpose of the Or-BAC model.

The central entity in Or-BAC is the entity $Organization$. Intuitively, an organization can be seen as any entity that is responsible for managing a given access control policy. Hence, hospitals or companies are organizations. A more concrete security component such as a firewall may be also viewed as an organization managing a network access control policy.

In Or-BAC, instead of defining security rules that directly apply to subject, action and object as in Rule-BAC, the access control policy is defined at the "organizational" level. For this purpose, subject, action and object are respectively abstracted into role, activity and view. A *view* corresponds to a set of objects to which the same security rules apply. An *activity* is similarly defined but for regrouping actions. Finally, permissions and prohibitions only apply in specific $contexts$. Examples of context may be *Night*, *Working-Hours* or *Emergency*.

Using these concepts of Or-BAC, $cond\_subject(s)$, $cond\_object(o)$ and $cond\_action(\alpha)$ respectively correspond to conditions specifying that, in a given organization, a subject is empowered in a role, an object is used in a view and an action implements a given activity. Finally, $constraint(s, a, o)$ is modelled as a condition that specifies that a subject performs an action on an object in a given $context$. For instance, the rule "in hospital H, a physician is permitted to consult his or her patient's medical record" may be represented by a rule having the following form:

- $condition \rightarrow Permission(s, a, o)$
  where, $condition$ is the following formula:
  $Empower(H, s, physician) \wedge Use(H, o, medical\_record) \wedge$
  $Consider(H, a, consult) \wedge Hold(H, s, a, o, attending\_physician)$

However, this is not exactly the way an access control policy is specified in the Or-BAC model. Actually, in Or-BAC, the access control policy does not directly apply to subject, action and object. Instead, the access control policy is specified using the organizational access control predicates *O-Permission* and *O-Prohibition*. For instance, the fact
$O\text{-}Permission(H, physician, consult, medical\_record, Attending\_physician)$
specifies that "hospital H grants to role *physician* the permission to consult $medical\_record$ in context $attending\_physician$".

Permission (resp. *Prohibition*) that applies to subject, action and object are then logically derived from organizational permissions (resp. prohibitions). For instance, John will obtain the permission to read a given object o if H empowers

John in role physician, *read* is an action that implements activity *consult*, *o* is an object used in view *medical_record* and John performs the action read on object *o* in the context *Attending_physician*.

Finally, we consider hierarchies of roles, but also of views, activities and contexts [10]. We associate these hierarchies with both inheritance of *O-Permission* and *O-Prohibition*. For instance, we can consider a role *suspended_physician* and specify that *suspended_physician* is a sub-role of *physician*. Thus, the role *suspended_physician* will inherit from the role *physician* the permission to consult *medical_record* in context *attending_physician*. This may lead to a conflict if a *suspended_physician* is explicitly prohibited to consult medical records. Thus, we now present how to manage conflicts in the Or-BAC model.

4.2 Principles of conflict management in Or-BAC

As in Prioritized Rule-BAC, we suggest managing conflicts in Or-BAC using priorities. We thus obtain the Prioritized Or-BAC model (see section 4.3 below for a formal presentation of this model). In Prioritized Or-BAC, priorities are assigned to *O-Permission* and *O-Prohibition*.

Managing *actual* conflicts in Prioritized Or-BAC is quite similar to Prioritized Rule-BAC and may be checked in polynomial time. However, compared with Rule-BAC, there is a major difference in the way we suggest managing redundant rules and potential conflicts in Prioritized Or-BAC.

Let us first explain our approach for potential conflict management. Since a subject can potentially be empowered in two different roles, an object can be used in two different views, an action can implement two different activities and two different contexts can be satisfied simultaneously (for instance take contexts *Night* and *Emergency*), every pair of *O-Permission* and *O-Prohibition* may be potentially conflicting. Thus, our approach to eliminate such potential conflict consists in specifying *separation constraints*. For instance, if a separation constraint exists between roles r_1 and r_2, then a subject cannot be empowered into both roles r_1 and r_2. As a consequence, if a given *O-Permission* is assigned to role r_1 and a given *O-Prohibition* is assigned to role r_2, then these *O-Permission* and *O-Prohibition* cannot generate a conflict. Similarly, we can specify separation constraints between views, activities and contexts.

If we compare to the Prioritized Rule-BAC model, using separation constraints provide a way to detect unrelated access control rules. However, whereas recognizing unrelated rules is generally undecidable, checking separation constraints is tractable and computable in polynomial time. As a consequence, we shall show that checking potential conflict in Prioritized Or-BAC is also tractable in polynomial time.

Let us now turn to redundant rule management. As in Prioritized Rule-BAC, the approach is based on exception detection. However our approach in Or-BAC is based on the inheritance hierarchy between roles, activities, views and contexts. More precisely, a given *O-Authorization* (that is an *O-Permission* or *O-Prohibition*) A_1 is an exception to another *O-Authorization* A_2, if A_2 is “higher” in the hierarchy than A_1. We shall show that using this approach to recognize exceptions is tractable in polynomial time. Thus, redundant rule management is also tractable in polynomial time in Prioritized Or-BAC.

Let us now present our formalization of the prioritized Or-BAC model.

4.3 The prioritized Or-BAC model

4.3.1 Modelling the organization components

The Prioritized Or-BAC model is built on top of the Prioritized Rule-BAC model defined in section 3. Thus, we consider Π a finite set of priorities. Π is associated with a partial order relation $\prec$.

To model the organization components of Prioritized Or-BAC, we then introduce the following domain predicates:

- $Empower(org, s, r)$: org empowers subject s in role r. Example: $Empower(H, John, physician)$ means that "Hospital H empowers $John$ in role $physician$."
- $Use(org, o, v)$: org uses object o in view v. Example: $Use(H, Doc\_31, medical\_record)$ means that "Hospital H uses object Doc_31 into view $medical\_record$."
- $Consider(org, \alpha, a)$: org considers that action α implements the activity a. Example: $Consider(H, acroread, consult)$ means that "Hospital H considers that action $acroread$ implements the activity $consult$."
- $Hold(org, s, \alpha, o, c)$: within organization org, context c holds between subject s, action α and object o. Example: $Hold(H, s, \alpha, o, attending\_physician)$ means that "within hospital H context $attending\_physician$ holds between subject s, action a and object o."
- $sub\_role(org, r_1, r_2)$: in organization org, role r_1 is a sub-role of r_2.

  We also use predicates $sub\_view(org, r_1, r_2)$, $sub\_activity(org, r_1, r_2)$, $sub\_context(org, r_1, r_2)$ to respectively specify hierarchies of views, activities and contexts. We assume that these hierarchies correspond to partial order relations.
- $seperated\_role(org, r_1, r_2)$: in organization org, role r_1 is separated from role r_2, i.e. a given subject cannot be empowered in both roles r_1 and r_2 (see associated constraints below). We assume that separation predicates correspond to anti-reflexive and symmetric relations.

  Similar predicates $seperated\_view(org, r_1, r_2)$, $separated\_activity(org, r_1, r_2)$ and $separated\_context(org, r_1, r_2)$ are introduced to respectively specify separations of views, activities and contexts.

All other predicates used in Prioritized Rule-BAC are also available in Prioritized Or-BAC. Regarding definition 2, we assume that $RDOM$ includes a set of context definition rules. For instance:

- $record\_patient(o, p) \wedge patient\_physician(p, s) \wedge action(a)$
  $\rightarrow Hold(H, s, a, o, Attending\_physician)$

This rule defines the context “Attending_physician” in hospital H as follows: a subject s performs the action a on object o in the context “Attending_physician” if o is a record corresponding to a patient of subject s.

- $record\_patient(o, p) \wedge urgent\_case(p) \wedge subject(s) \wedge action(a)$
  $\rightarrow Hold(H, s, a, o, Emergency)$
  This rule defines the context “Emergency” in H.

More detailed definition and use of contexts in Or-BAC can be found in [12].

Regarding definition 3, we assume that Cte includes the following separation constraints:

- $separated\_role(org, r_1, r_2) \wedge Empower(org, s, r_1) \wedge Empower(org, s, r_2)$
  $\rightarrow error()$
  Similar constraints are defined for predicates $separated\_view$ and $separated\_activity$ by replacing predicates $Empower$ by predicates Use (resp. $Consider$).
- $separated\_context(org, c_1, c_2) \wedge Hold(org, s, a, o, c_1) \wedge Hold(org, s, a, o, c_2)$
  $\rightarrow error()$

4.3.2 Policy definition

To define the access control policy, we introduce the following organizational access control predicates:

- $O\text{-}Permission(org, r, a, v, c, p)$: means that in organization org, role r is granted permission to perform activity a on view v within context c. Moreover, this organizational permission is assigned priority p.
- $O\text{-}Prohibition(org, r, a, v, c, p)$: similar to predicate $O\text{-}Permission$ but for organizational prohibition.
- $O\text{-}Authorization(org, r, a, v, c, p)$: corresponds to an $O\text{-}Permission$ or $O\text{-}Prohibition$.
- $DO\text{-}Permission(orgr, a, v, c, p)$: similar to predicate $O\text{-}Permission$ but this is a *derived* organizational permission through hierarchies of inheritance.
- $DO\text{-}Prohibition(orgr, a, v, c, p)$: similar to predicate $O\text{-}Prohibition$ but this is a *derived* organizational prohibition.

In Prioritized Or-BAC, a prioritized access control policy is defined as follows:

Definition 18: A prioritized access control policy $PACP$ includes the following set of facts and rules:

- Facts having the form $O\text{-}Permission(org, s, a, r, c, p)$ or $O\text{-}Prohibition(org, s, a, r, c, p)$. For instance the fact:
  $O\text{-}Permission(H, physician, consult, med\_record, Attending\_physician, p_1)$
  specifies that, in organization H, the role *physician* is permitted to *consult* the view *med_record* in context *Attending_physician*. This permission is assigned priority p_1.

The following rule specifies inherited permissions through the role hierarchy:

- $\mathit{DO\text{-}Permission}(org, r_2, a, v, c, p) \wedge \mathit{sub\_role}(org, r_1, r_2)$
  $\rightarrow \mathit{DO\text{-}Permission}(org, r_1, a, v, c, p)$

A similar rule applies to inheritance of prohibitions and predicate *DO-Prohibition* is used for this purpose. The hierarchy on activities, views and contexts, are similarly associated with the inheritance mechanism that is modelled in the previous rules.

The following rules say that an organizational permission is also a derived permission and a (positive) authorization:

- $\mathit{O\text{-}Permission}(org, r, a, v, c, p) \rightarrow \mathit{DO\text{-}Permission}(org, r, a, v, c, p)$
- $\mathit{O\text{-}Permission}(org, r, a, v, c, p) \rightarrow \mathit{O\text{-}Authorization}(org, r, a, v, c, p)$

There are similar rules to specify that an organizational prohibition is also a derived prohibition and a (negative) authorization

There is the following general rule to derive instances of Prima Facie *Permission* from Derived Organizational *DO-Permission*:

- $\mathit{DO\text{-}Permission}(org, r, a, v, c, p) \wedge$
  $\mathit{Empower}(org, s, r) \wedge \mathit{Use}(org, o, v) \wedge \mathit{Consider}(org, \alpha, a) \wedge \mathit{Hold}(org, s, a, o, c)$
  $\rightarrow \mathit{Permission}(s, \alpha, o, p)$ (GR1)

There is another general rule called (GR2) to derive instances of Prima Facie *Prohibition* from Derived Organizational *DO-Prohibition*.

Definition of the global conflict resolution strategy GCRS is the same as the one of definition 12 used in Prioritized Rule-BAC.

4.4 Redundant rules management

As in the Prioritized Rule-BAC model, redundant rules management is based on the notion of exception to a general authorization. For this purpose, let $A_i = \mathit{O\text{-}Authorization}(org, r_i, a_i, v_i, c_i, p_i)$ and $A_j = \mathit{O\text{-}Authorization}(org, r_j, a_j, v_j, c_j, p_j)$ be two (positive or negative) authorizations.

Definition 19: Authorization A_i is a strict exception to authorization A_j if the following condition holds:

$$\mathit{sub\text{-}role}(org, r_i, r_j) \wedge \mathit{sub\text{-}activity}(org, a_i, a_j) \wedge$$
$$\mathit{sub\text{-}view}(org, v_i, v_j) \wedge \mathit{sub\text{-}context}(org, c_i, c_j) \wedge$$
$$\neg(r_i = r_j \wedge a_i = a_j \wedge v_i = v_j \wedge c_i = c_j)$$

The main difference with Prioritized Rule-BAC is that we can now prove the following theorem:

Theorem 7: Checking that authorization A_i is an exception to authorization A_j is tractable in polynomial time.

Proof: Trivial. Checking that authorization A_i is an exception to authorization A_j corresponds to checking conditions in a stratified Datalog program, so it is tractable in polynomial time. □

Thus, checking the non-redundancy condition defined in section 3.4 is also tractable in polynomial time.

4.5 Potential conflict detection

Definition 20 (Potential conflict condition): A potential conflict exists if the following condition is satisfied:

$$\begin{aligned}&\textit{DO-Permission}(org, r_i, a_i, v_i, c_i, p_i) \wedge \textit{DO-Prohibition}(org, r_j, a_j, v_j, c_j, p_j) \wedge\\ &\neg(\textit{separated-role}(org, r_i, r_j)) \wedge \neg(\textit{separated-activity}(org, a_i, a_j)) \wedge\\ &\neg(\textit{separated-view}(org, v_i, v_j)) \wedge \neg(\textit{separated-context}(org, c_i, c_j)) \wedge\\ &\neg \textit{Solved-Conflict}(org, r_i, a_i, v_i, c_i, p_i, r_j, a_j, v_j, c_j, p_j)\end{aligned}$$

where *Solved-Conflict* predicate is defined by the following rule:

$$\begin{aligned}&\textit{DO-Permission}(org, r_i, a_i, v_i, c_i, p_i) \wedge \textit{DO-Prohibition}(org, r_j, a_j, v_j, c_j, p_j) \wedge\\ &((\textit{DO-Prohibition}(org, r, a, v, c, p) \wedge (p_i \prec p))\\ &\vee(\textit{DO-Permission}(org, r, a, v, c, p) \wedge (p_j \prec p))) \wedge\\ &(r = r_i \vee r = r_j) \wedge (a = a_i \vee a = a_j) \wedge (v = v_i) \vee v = v_j) \wedge (c = c_i \vee c = c_j)\\ &\quad \rightarrow \textit{Solved-Conflict}(org, r_i, a_i, v_i, c_i, p_i, r_j, a_j, v_j, c_j, p_j)\end{aligned}$$

This is a quite complex condition, however using this condition of potential conflict, we can prove theorem 5 stated in section 3.5.

Proof: Let us assume that there is an actual conflict. It follows from definition 12 that:

$$\begin{aligned}&\textit{Permission}(s, a, o, p_i) \wedge \neg \textit{Higher-Prohibition}(s, a, o, p_i) \wedge\\ &\textit{Prohibition}(s, a, o, p_j) \wedge \neg \textit{Higher-Permission}(s, a, o, p_j) \qquad (1)\end{aligned}$$

Since in Or-BAC, the only way to derive an actual permission and an actual prohibition is by respectively applying GR1 and GR2, we can conclude that:

$$\textit{DO-Permission}(org, r_i, a_i, v_i, c_i, p_i) \wedge \textit{DO-Prohibition}(org, r_j, a_j, v_j, c_j, p_j)$$

and that we have also:

$$\textit{Empower}(org, s, r_i) \wedge \textit{Empower}(org, s, r_j)$$

and thus we can conclude that $\neg \textit{separated-role}(org, r_i, r_j)$ else *error* could be derived (which is in contradiction with the assumption that no constraint is violated).

We can similarly derive other separation conditions on activity, view and context that appear in definition 20.

Finally, if we assume that:

$$\textit{Solved-Conflict}(org, r_i, a_i, v_i, c_i, p_i, r_j, a_j, v_j, c_j, p_j)$$

then we can derive that there is some priority level p such that:

$$\begin{aligned}&(\textit{Prohibition}(s, a, o, p) \wedge \textit{Higher-Prohibition}(s, a, o, p_1)) \vee\\ &(\textit{Permission}(s, a, o, p) \wedge \textit{Higher-Permission}(s, a, o, p_2))\end{aligned}$$

but this is in contradiction that there is an actual conflict. □

By the way, we can and also prove the following theorem:

Theorem 8: Checking the above condition of potential conflict is decidable in polynomial time.

Proof: Trivial. Proof is similar to theorem 7. □

This last condition is very useful because, using the contraposition of theorem 5, we can derive that if there is no potential conflict according to this condition, then we can guarantee that we can never derive any actual conflict. In particular, this condition guarantees that we can insert new subjects, objects or actions without creating actual conflicts.

4.6 Example

We use the same example of access control policy as the one presented in section 3.6, and we show how to model it in Prioritized Or-BAC. Here, we assume that Π is $\{p1, p2, p3, p4, p5\}$. In Prioritized Or-BAC, access control rules are expressed as follows (we assume that they apply to a given hospital H):

- $O\text{-}Prohibition(H, nurse, consult, medical\_record, Default, p1)$
  (rule $R1$ associated with priority $p1$. $Default$ is a context that is always true for every subject, action and object).
- $O\text{-}Permission(H, nurse, consult, medical\_record, Emergency, p2)$
  (rule $R2$ associated with priority $p2$)
- $O\text{-}Permission(H, physician, consult, medical\_record, Attending\_physician, p3)$
  (rule $R3$ associated with priority $p3$)
- $O\text{-}Prohibition(H, suspended\_physician, consult, medical\_record, Default, p4)$
  (rule $R4$ associated with priority $p4$)
- $O\text{-}Prohibition(H, suspended\_nurse, consult, medical\_record, Default, p5)$
  (rule $R5$ associated with priority $p5$)

We also consider the following inheritance hierarchies and separation constraints:

- $sub\_role(H, suspended\_physician, physician) \wedge$
  $sub\_role(H, suspended\_nurse, nurse)$
- $sub\_context(H, Attending\_physician, Default) \wedge$
  $sub\_context(H, Emergency, Default)$
- $separated\_role(H, nurse, physician) \wedge$
  $separated\_role(H, suspended\_nurse, physician) \wedge$
  $separated\_role(H, nurse, suspended\_physician) \wedge$
  $separated\_role(H, suspended\_nurse, suspended\_physician)$

Now, we can check that rule $R1$ has two exceptions, namely $R2$ and $R5$. Thus, using the no redundancy condition, we can derive that $p1 \prec p2$ and $p1 \prec p5$. There is no other exception in our example (in particular, one can check that $R4$ is not an exception to $R3$).

Then, if we apply the separation constraints and the potential conflict condition, we can check that there are two potential conflicts: the first one between rules $R2$ and $R5$ and the second between $R3$ and $R4$. To solve this conflict, we have to state priorities between $p2$ and $p5$ and between $p3$ and $p4$, for instance $p2 \prec p5$ and $p3 \prec p4$. Thus, in this case, we obtain $p1 \prec p2 \prec p5$ and $p3 \prec p4$. Now, the access control policy is free of potential conflict, and one can guarantee that it is not possible to derive actual conflict when applying this policy. Notice that it was not necessary to totally order the set of priorities; a partial order is sufficient and is easier to manage.

Finally, as suggested in section 3.7, one can easily specify automatic assignment strategies in Prioritized Or-BAC. For instance, we could say that security rules that apply to the emergency context have higher priority than other rules. This assignment strategy can be expressed as follows:

$$\begin{aligned}&\textit{O-Autorization}(org, r_i, a_i, v_i, emergency, p_i) \wedge\\&\textit{O-Autorization}(org, r_j, a_j, v_j, c, p_j) \wedge \neg(c = emergency)\\&\qquad \rightarrow (p_j \prec p_i)\end{aligned}$$

Applying this rule, we can derive that $p2$ has higher priority than all other priorities. This can be used to automatically solve the potential conflict between $R2$ and $R5$. Due to space limitation, we do not further develop how to specify automatic assignment strategies.

5 Comparison with related works

We choose to deal in this paper with models termed as Rule-BAC models. We choose to compare them with a more structured and richer models like Or-BAC as we argue that Rule-BAC model needs to be enhanced to circumvent many problems that are consequences of its lack of structure. Or-BAC model, as we show in this paper, is free from redundant rules and potential conflicts detection undecidability which is not the case of Rule-BAC model. The resolution of conflicts is done at the "organizational" level so that we give assurance that no actual conflict occurs between triples $\langle subject, action, object\rangle$.

In this section, we state the results of a more global look at other works in this domain. In the same way, recent models consider prohibitions and are of course faced to conflicts. This is the case of the model suggested in [5]. In this paper, Bertino, Jajodia and Samarati suggest an authorization mechanism that enables multiple access control policies to be supported. The mechanism enforces a general authorization model that manages both positive and negative authorizations. It also distinguishes between weak and strong authorizations. A strong authorization overrides a weak authorization whereas a strong authorization cannot be overridden. In this model, only conflicts between weak authorizations are manageable and, in this case, the authors propose an approach to resolve conflicts.

Another approach is presented in [15]. In this model, the strategy to manage conflicts is "hard-coded" in the sense that there is no clear separation between the strategy for conflict management and the remainder of the policy specification. Moreover, the authors only consider four possible conflict management strategies: No conflict (in this case, a conflict is viewed as a constraint violation), prohibitions take precedence, permissions take precedence and nothing takes precedence (means that there is actually no conflict resolution). Thus, from the point of view of conflict management, this approach is less flexible than Prioritized Or-BAC.

A more recent approach is suggested in [3]. This model specifies an access control policy as a set of logical rules expressed in a language that is quite similar to [16]. It suggests managing conflicts using the concept of *conflict resolution policy* (crp). This idea is quite similar to our concept of conflict resolution strategy. However, definition of a crp is fully separated from the remainder of the policy specification and is only used to provide semantics to the suggested model. In our approach, a CRS is used to assign priority levels to permissions and prohibitions. This is used to model the concept of prioritized policy as a single logical theory that includes both the policy specification and the CRS. Thus, we obtain a more integrated model.

In [7], a security policy is modelled using modal logic, permissions, prohibitions and obligations being represented using deontic modalities. This provides a richer model in which it is possible to specify, for instance, disjunctive obligations or conjunctive prohibitions. However, this paper only suggests managing conflicts using priority between roles. In [9], the approach is refined and the concept of strategy to manage conflicts is introduced. However, this strategy is used to define priority between roles and its specification is separated from the remainder of the policy specification. Thus, our approach is more integrated and provides means to specify more flexible strategies. Moreover, the complexity of reasoning with the strategies is not addressed in [7].

Finally, [2] suggests an approach based on possibilistic logic to handle conflicts in prioritized security policies. The priority is implicitly derived from the format of rules. This is used to effectively construct a stratified theory in which conflicts are solved. However, this strategy is not always effective to solve every conflict and must be refined to handle unsolved conflict situations.

6 Conclusion

In this paper, we show the advantage of managing conflict in an access control policy modelled in Or-BAC. Since a policy in Or-BAC is defined at an organizational level (i.e. independently of actual implementation of subjects, objects and actions in the system), we suggest managing conflicts at the organizational level. Our approach is based on defining conflict resolution strategy (CRS) that is used to assign priority levels to organizational permissions or prohibitions. Two different situations may arise when using a CRS: (1) Redundant rules may exist and (2) potential conflict may arise.

We show that these two problems are tractable in polynomial time in the Or-BAC model unlike models using an equivalent logical approach to specify access control rules but using a less structured framework, say Rule-BAC models. Moreover, a tool, called MotOrBAC has been designed and implemented to specify an Or-BAC policy and manage conflicts (see http://www.orbac.org for more information).

There are several possible extensions to this work. First Or-BAC also includes the possibility to specify obligations. However, Since conflicts can also occur between obligation and prohibition [6,8] , managing such conflicts represent further work that remains to be done.

In this paper, we only investigate the problem of conflicts within a single organization. We are also applying our approach to control interoperability of organizations managing different security policies. Our proposal is well suited to detect and manage conflicts that occurs when these organizations want to interoperate.

Finally, in [11], AdOr-BAC, an administration model for Or-BAC is defined and implemented. It would be interested to extend our approach to manage conflicts between administration rules. In particular, AdOr-BAC includes the possibility to specify delegation rules that may generate conflicts with other security rules. Managing these conflicts represents further work that remains to be done.

References

[1] Y. Bai and V. Varadharajan. A Logical Based Approach for the Transformation of Authorization Policies. In *Proc. of the 10th Computer Security Foundations Workshop*, Rockport, Massachusetts, 1997.

[2] S. Benferhat, R. El Baida, and F. Cuppens. A Stratification-Based Approach for Handling Conflicts in Access Control. In *8th ACM Symposium on Access Control Models and Technologies (SACMAT'03)*, Lake Come, Italy, June 2003.

[3] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A Logical Framework for Reasoning about Access Control Models. *ACM Transactions on Information and System Security*, 6(1), February 2003.

[4] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. On comparing the Expressing Power Of Access Control Model. In *Foundations of Computer Security (FCS'04)*, Turku, Finland, July 2004.

[5] E. Bertino, S. Jajodia, and P. Samarati. Supporting Multiple Access Control Policies in Database Systems. In *IEEE Symposium on Security and Privacy*, Oakland, 1996.

[6] C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera. Obligation Monitoring in Policy Management. In *Third International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, 2004.

[7] L. Cholvy and F. Cuppens. Analyzing Consistency of Security Policies. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 1997.

[8] J. Chomicki and J. Lobo. A Logic Programming Approach to Conflict Resolution in Policy Management. In *7th International Conference on Principles of Knowledge Representation and Reasoning (KR'2000)*, Breckenridge, Colorado, April 2000.

[9] F. Cuppens, L. Cholvy, C. Saurel, and J. Carrère. Merging regulations: analysis of a practical example. *International Journal of Intelligent Systems*, 16(11), November 2001.

[10] F. Cuppens, N. Cuppens-Boulahia, and A. Miège. Inheritance hierarchies in the Or-BAC Model and application in a network environment. In *Second Foundations of Computer Security Workshop (FCS'04)*, Turku, Finland, July 2004.

[11] F. Cuppens and A. Miège. Administration Model for Or-BAC. In *International Federated Conferences (OTM'03), Workshop on Metadata for Security*, Catania, Sicily, Italy, November 2003.

[12] F. Cuppens and A. Miège. Modelling contexts in the Or-BAC model. In *19th Annual Computer Security Applications Conference*, Las Vegas, December 2003.

[13] J. Y. Halpern et V. Weissman. Using First-order Logic to Reason about Policies. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)*, Pacific Grove, California, USA, July 2003.

[14] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in Operating Systems. *Communication of the ACM*, 19(8):461–471, August 1976.

[15] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible Support for Multiple Access Control Policies. *ACM Transactions on Database Systems*, 26(2), June 2001.

[16] S. Jajodia, S. Samarati, and V. S. Subrahmanian. A logical Language for Expressing Authorizations. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 1997.

[17] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Come, Italy, June 2003.

[18] Jeffrey D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1,2. Computer Science Press, edit. W. H. Freeman, 1989.